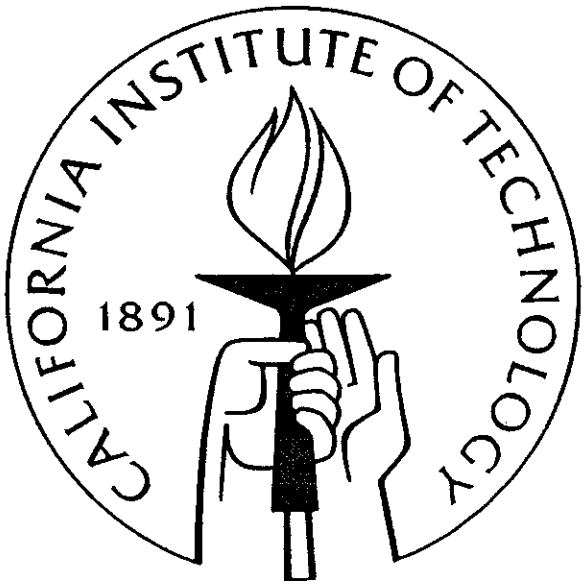


Recurrence-Based Reductions for Inclusion and Exclusion Algorithms Applied to #P Problems

Eric Bax

Computer Science Department
California Institute of Technology

Caltech-CS-TR-96-01

Recurrence-Based Reductions for Inclusion and Exclusion Algorithms Applied to #P Problems

Eric Bax*

January 23, 1996

Abstract

There are inclusion and exclusion algorithms for many #P problems. By imposing a hierarchy on the inclusion and exclusion formula's terms, we develop general reductions for inclusion and exclusion algorithms. We outline sufficient steps to tailor the general reductions to specific problems, and we illustrate this process by applying it to algorithms for several #P problems. We test the reductions on the problem of counting Hamiltonian paths. Within the framework of the general reductions, we develop the concepts of zero sets and vestigial elements, and we present problem decomposition strategies. Also, we show how the reductions can be applied to algorithms that give approximate solutions to #P problems.

Key words algorithms, combinatorial problems, parallel algorithms.
AMS subject classifications 05,68

*Supported by an NSF fellowship. Computer Science Department, California Institute of Technology 256-80, Pasadena, California, 91125 (eric@csvax.caltech.edu).

Acknowledgements

I thank three great teachers at Furman University: Dr. Douglas Rall for introducing me to combinatorics and to research, Dr. P. M. Cook for listening and guidance, and Dr. Hayden Porter for teaching me two concepts that are central to this work – recursion and abstraction. Thanks to Dr. Andras Recski at MTKI in Budapest for teaching me when I was a student there, and also for taking the time to read my papers and point me in useful directions when I walked into his office out of the blue several years later. I thank Dr. Joel Franklin for showing an interest in this work, for listening patiently as I tried to explain it, and for supplying just the right information at just the right time. I thank my advisor Dr. Yaser S. Abu-Mostafa for his support, advice, and example.

Contents

List of Figures

1 Introduction

1.1 #P Problems

#P problems are the counting problems associated with existence problems that are in NP. For example, determining whether or not a graph contains a Hamiltonian cycle is an NP problem. So counting the Hamiltonian cycles in a graph is a #P problem. The Hamiltonian cycle existence problem is NP-complete [7], meaning that every NP problem is polynomially reducible to Hamiltonian cycle existence. Likewise, the counting problem is #P-complete [20], meaning that every #P problem is polynomially reducible to counting Hamiltonian cycles.

Clearly, if a population can be counted in polynomial time, then the existence of a member of the population can be determined in polynomial time. So polynomial time solutions for #P problems are also solutions for the associated NP problems.

For example, given a graph G with n vertices and adjacency matrix A, consider the problem of determining whether or not there is a length n walk from vertex 1 back to itself. Since a candidate walk can be checked for existence in G in time proportional to n, the problem is in NP. Hence, counting all length n 1-1 walks is in #P. The counting problem can be solved in polynomial time since $[A^n]_{11}$ is its solution. So the existence problem can also be solved in polynomial time by computing $[A^n]_{11}$. If it is nonzero, then there is a length n 1-1 walk; otherwise there is not.

The NP problems associated with #P-complete problems need not be NP-complete (unless P=NP). Consider the NP problem of determining whether or not a given vertex is on a cycle. This problem is in P because there is a cycle on vertex i if and only if $[A^m]_{ii} > 0$ for some $m \leq n$. But counting the cycles on a vertex is #P-complete [20].

For definitions and in-depth treatment of P, NP, NP-completess, #P, and #P-completeness, refer to [4].

Algorithms for #P problems return a population size, not a list of the population's members. When the population size is exponential in the size of the input, a listing algorithm must take at least exponential time. Designers of listing algorithms try to achieve minimum computation per member of the population, while designers of counting algorithms try to determine the population size without enumerating the individual members of the population.

For example, consider the population of cycles in a graph. Tarjan's cycle listing algorithm [18] has time complexity $O(n|E||C|)$, where n, $|E|$, and $|C|$ are the numbers of vertices, edges, and cycles, respectively. There is a cycle counting algorithm [2] with time complexity $O(2^n poly(n))$. Since there may be $O(n!)$ cycles, the counting algorithm is more efficient for simply determining the number of cycles.

1.2 Inclusion and Exclusion Algorithms

To compare straightforward computation, dynamic programming, and inclusion and exclusion algorithms, consider the calculation of the permanent of an $n \times n$ matrix A. Valiant [19] has shown this problem to be #P-complete. By definition:

$$perA = \sum_{j_1 \ldots j_n} a_{1j_1} \ldots a_{nj_n} \tag{1}$$

where $j_1 \ldots j_n$ is a permutation of $1 \ldots n$. Straightforward computation requires $O(n!\ poly(n))$ time and $O(poly(n))$ space.

Examine a dynamic programming algorithm for the same problem:

initially $T(\emptyset) = 1$
for $k = 1$ to n
  for every $S \subseteq \{1 \ldots n\}$ with $|S| = k$
    $T(S) := \sum_{s \in S} T(S - \{s\}) a_{ks}$

Upon termination, $T(S) = \sum_{j_1 \ldots j_k} a_{1j_1} \ldots a_{kj_k}$, where $k = |S|$, and $j_1 \ldots j_k$ is a permutation of the elements of S. Hence, $T(\{1 \ldots n\})$ is the permanent. By gathering terms, dynamic programming reduces the time complexity to $O(2^n)$. However, the space complexity increases quite a bit. To compute the values of $T(S)$ for the sets S with k elements, the algorithm must store the $C(n, k-1)$ values $T(S)$ for all S with $k-1$ elements. This requires $C(n,k) + C(n, k-1)$ storage locations. So the algorithm has space complexity $O(C(n, \lceil \frac{n}{2} \rceil)) = O(\frac{1}{\sqrt{n}} 2^n)$.

Now examine the inclusion and exclusion algorithm by Ryser [16]:

$$perA = \sum_{S \subseteq \{1 \ldots n\}} (-1)^{|S|} (\prod_i \sum_{j \notin S} a_{ij}) \tag{2}$$

This algorithm will be fully explained later in this paper; for now just consider its time and space requirements. The parenthesized term is a product of row sums, which can be computed in $O(n^2)$ time and $O(n^2)$ space. There is one such term for every $S \subseteq \{1 \ldots n\}$, so the total time required is $O(2^n poly(n))$. Since earlier terms are not used to compute later terms, the same space may be reused to compute each term. So the inclusion and exclusion algorithm requires $O(poly(n))$ space, compared to $O(2^n)$ for the dynamic programming algorithm.

The inclusion and exclusion algorithm can be implemented on a message-passing multicomputer as follows:

1. Load the problem into each processor.
2. Assign each processor a subset of the terms.

3. Have each processor independently compute its terms and sum.

4. Sum over all processors to find the permanent.

The only communication required is to load the problem initially and to sum over all processors at the end. The terms of the inclusion and exclusion algorithm can be computed independently, but the terms of the dynamic programming algorithm are highly interdependent. Consequently, much more synchronization waiting and message passing are necessary to implement the dynamic programming algorithm on a parallel computer.

1.3 Inclusion and Exclusion Algorithms for #P Problems

There are inclusion and exclusion algorithms for many #P problems, including counting vertex colorings [21], computing the matrix permanent [16], counting Hamiltonian paths, counting feasible solutions for sequencing and bin packing [8], and counting satisfying assignments for conjunctive normal form (CNF) expressions [10]. The primary example in this paper is an inclusion and exclusion algorithm to count Hamiltonian paths. As results are developed, they are generalized and then applied to counting vertex colorings, counting CNF satisfiers, and computing the matrix permanent.

2 Inclusion and Exclusion

2.1 Definitions and Notation

Given a universal set U, sets $B_1 \ldots B_n$, and some means of counting elements in intersections of set complements $\overline{B_1} \ldots \overline{B_n}$, inclusion and exclusion counts the elements in $B_1 \cap \ldots \cap B_n$.

Definition 1 $N(R,S) \equiv |(\bigcap_{i \in R} B_i) \cap (\bigcap_{j \in S} \overline{B_j})|$.

$N(R,S)$ counts the members of U that are in every set indexed by R and that are not in any set indexed by S. (This is an extension of Karp's $N(S)$ [8].) Define $N(\emptyset,\emptyset)$ to be $|U|$.

For an illustration and examples of $N(R,S)$, see Figures 1 and 2.

Theorem 1 (Recursion)

$$\forall r \in R, N(R,S) = N(R-\{r\},S) - N(R-\{r\},S \cup \{r\}) \tag{3}$$

Proof. Let $\hat{U} \equiv (\bigcap_{i \in R-\{r\}} B_i) \cap (\bigcap_{j \in S} \overline{B_j})$. Then we can rewrite (3):

$$|\hat{U} \cap B_r| = |\hat{U}| - |\hat{U} \cap \overline{B_r}| \tag{4}$$

$$|\hat{U}| = |\hat{U} \cap B_r| + |\hat{U} \cap \overline{B_r}| \tag{5}$$

□

For an illustrated example of the recursion, see Figure 3.

Iterating the recursion to $|R|$ levels produces the formula:

$$N(R,S) = \sum_{A \subseteq R} (-1)^{|A|} N(\emptyset, S \cup A) \tag{6}$$

Applying this formula to $N(\{1 \ldots n\},\emptyset)$ gives the inclusion and exclusion formula:

$$N(\{1 \ldots n\},\emptyset) = \sum_{S \subseteq \{1 \ldots n\}} (-1)^{|S|} N(\emptyset, S) \tag{7}$$

$$|B_1 \cap \ldots \cap B_n| = \sum_{S \subseteq \{1,\ldots,n\}} (-1)^{|S|} |\bigcap_{k \in S} \overline{B_k}| \tag{8}$$

where $\bigcap_{k \in \emptyset} \overline{B_k} \equiv U$.

For a different proof of inclusion and exclusion, see [22]. A more general inclusion and exclusion result is given by H. J. Ryser in [16].

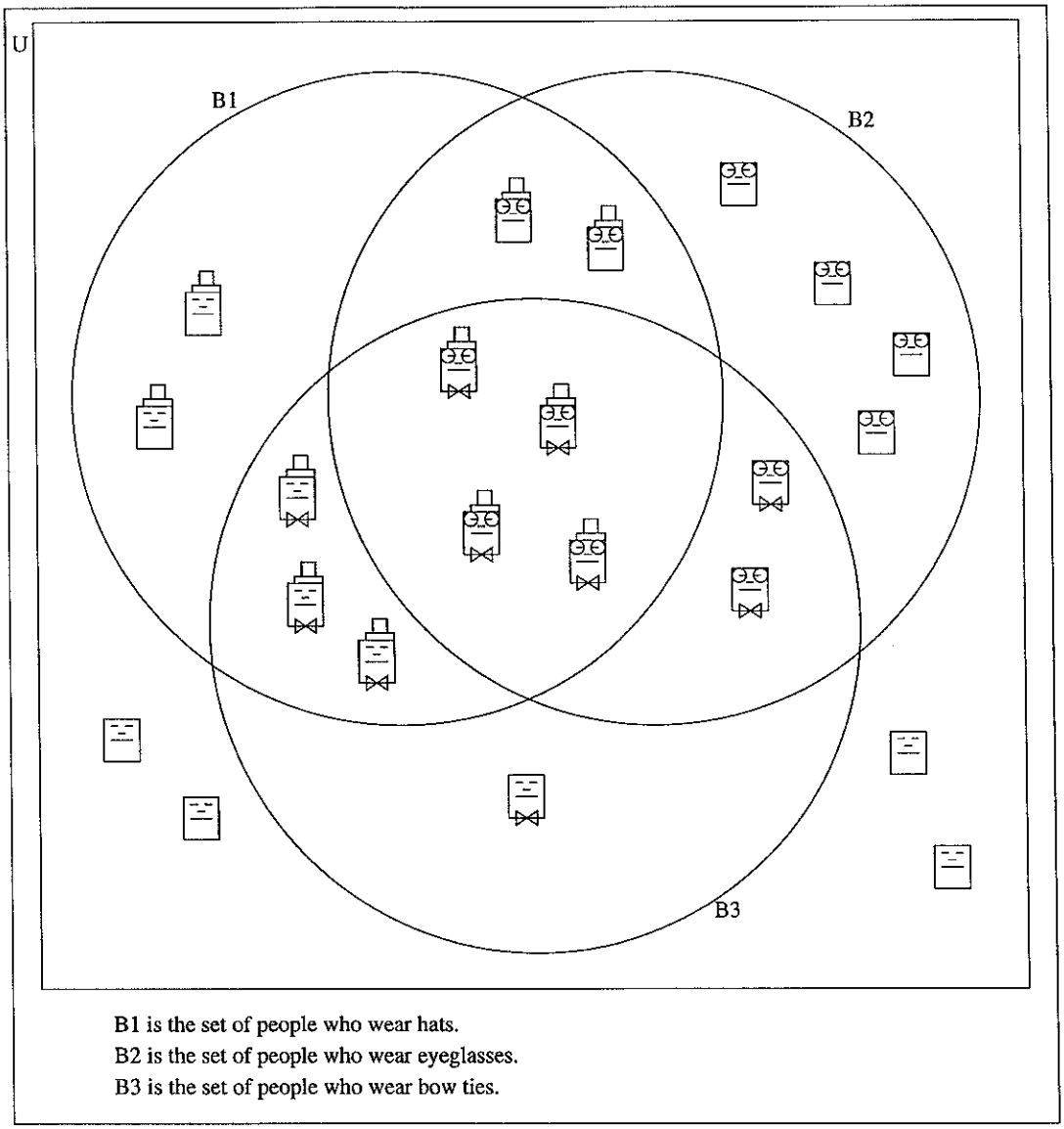
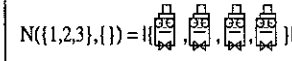


Figure 1: A Set System to Illustrate $N(R, S)$

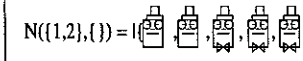
Examples of N(R,S):

N(R,{}):

N({1,2,3},{}) is the size of the intersection of B1, B2, and B3.
This is the number of people who wear hats, eyeglasses, and bow ties.

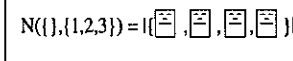


N({1,2},{}) is the size of the intersection of B1 with B2.
This is the number of people who wear hats and eyeglasses.

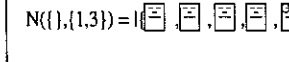


N({},S):

N({},{1,2,3}) is the size of the intersection of the complements of B1, B2, and B3.
This is the number of people who wear neither hats, nor eyeglasses, nor bow ties.



N({},{1,3}) is the size of the intersection of the complements of B1 and B3.
This is the number of people who wear neither hats nor bow ties.



N(R,S):

N({1},{2}) is the size of the intersection of B1 with the complement of B2.
This is the number of people who wear hats and do not wear eyeglasses.

N({1,3},{2}) is the size of the intersection of B1, B3, and the complement of B2.
This is the number of people who wear hats and bow ties and do not wear eyeglasses.

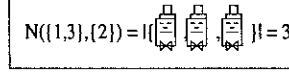


Figure 2: $N(R,S)$ Examples in the Illustrative Set System

The Recursion Formula N(R,S) = N(R-r,S) - N(R-r,Su{r}):

N({1,3},{2}) = N({1},{2}) - N({1},{2,3})
This is the recursion formula, with "split" element r=3.

In our set system, the equation says:
The number of people who wear hats and bow ties and do not wear eyeglasses
is the number of people who wear hats and do not wear eyeglasses,
minus the number of people who wear hats and wear neither eyeglasses nor bow ties.

Bounding: N(Q,T) is greater than or equal to N(R,S) if Q is a subset of R and T is a subset of S:

N({1},{}) is greater than or equal to N({1,2},{3}).
In our set system, this says:
The number of people who wear hats is greater than or equal to the number of people
who wear hats and eyeglasses and do not wear bow ties.

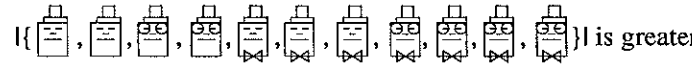
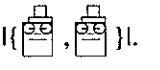

is greater than


Figure 3: Recursion and Bounding

2.2 Counting Hamiltonian Paths

To demonstrate the notation, consider the problem of counting s-t Hamiltonian paths in a directed graph with vertex set $V = \{s,t\} \cup \{1 \ldots n\}$. Let U be the set of length $n+1$ s-t walks in G. Let B_i be the set of walks that visit vertex i. Then $N(R,S)$ is the number of walks that visit every vertex in R, visit no vertex in S, and visit none, some, or all of the vertices that are neither in R nor S. $N(\{1 \ldots n\}, \emptyset)$ is the number of s-t Hamiltonian paths.

Use of the inclusion and exclusion formula requires a method to compute $N(\emptyset, S)$ for subsets S of $\{1 \ldots n\}$. Recall that adjacency matrix multiplication counts walks, i.e. if A is the adjacency matrix of G, then $[A^{n+1}]_{st}$ is the number of length $n+1$ s-t walks in G. Since $N(\emptyset, S)$ is the number of s-t walks that do not visit any vertex in S, $N(\emptyset, S)$ counts walks in the graph $G(V-S)$ formed from G by destroying all edges to and from vertices in S. So to compute $N(\emptyset, S)$, form $A(V-S)$ from A by zeroing all rows and columns corresponding to elements of S; then $N(\emptyset, S) = [A(S)^{n+1}]_{st}$. This method is explained with more detail in [1]. Another method to compute $N(\emptyset, S)$ is given in [8].

3 Theorems, Reductions, and the Counting Algorithm

3.1 $N(R,S)$ Bounding and Symmetry Theorems

Theorem 2 (Nonnegativity) $N(R,S) \geq 0$.

Proof. N(R,S) is the size of a set. □

Theorem 3 (Bounding) $\forall\ Q \subseteq R$ *and* $T \subseteq S$, $N(Q,T) \geq N(R,S)$.

Proof. The set counted by $N(R,S)$ is a subset of the set counted by $N(Q,T)$:

$$(\bigcap_{i\in R} B_i) \cap (\bigcap_{j\in S} \overline{B_j}) \tag{9}$$

$$= [(\bigcap_{i\in Q} B_i) \cap (\bigcap_{j\in T} \overline{B_j})] \cap [(\bigcap_{i\in R-Q} B_i) \cap (\bigcap_{j\in S-T} \overline{B_j})] \tag{10}$$

$$\subseteq (\bigcap_{i\in Q} B_i) \cap (\bigcap_{j\in T} \overline{B_j}) \tag{11}$$

□

For an illustration of bounding, see Figure 3.

Theorem 4 (Symmetry) *If* $\exists M \subseteq R$ *such that* $\forall\ A, B \subseteq M$ *with* $|A| = |B|$, $N(R-M, S\cup A) = N(R-M, S\cup B)$ *then:*

$$N(R,S) = \sum_{i=0}^{|M|} (-1)^i C(|M|,i) N(R-M, S\cup A_i) \tag{12}$$

where A_i *is any size* i *subset of* M.

Proof. Expand $N(R,S)$ by $|M|$ recursions (3), using the elements of M as the "split" elements r. Collect the resulting terms $N(R-M,\hat{S})$ according to $|\hat{S}|$. □

3.2 Reductions

3.2.1 R Subset Reduction

For $|T| = |S|$, the bounding theorem supplies bounds for $N(R,S)$ that increase in accuracy and computational expense as $|Q|$ increases. $|Q| = 0$ yields $N(\emptyset,S) \geq N(R,S)$. $|Q| = 1$ yields $N(\{r\},S) = N(\emptyset,S) - N(\emptyset, S\cup\{r\}) \geq N(R,S)$ for every $r \in R$.

By the nonnegativity and bounding theorems, $N(Q,S) \geq N(R,S) \geq 0$, so $N(Q,S) = 0$ implies $N(R,S) = 0$. In this case $N(R,S)$ need not be computed by (3), so $O(2^{|R|}poly(n))$ time is saved.

3.2.2 Symmetry Reduction

If we can find a set M meeting the conditions of the symmetry theorem, then we save $O((2^{|R|}-(|M|+1)2^{|R-M|})poly(n))$ computation time by using (12) instead of (3) to compute $N(R,S)$. If $|M|=1$ then the symmetry formula reduces to the recursion formula.

To clarify the use of the symmetry reduction, consider once again the problem of counting Hamiltonian paths. Define $G(V-S)$ to be the graph induced by vertex set $V-S$. To satisfy the conditions of the symmetry theorem, it is sufficient to find $M \subseteq R$ such that $\forall\, A,B \subseteq M$ with $|A|=|B|$, $G(V-(S\cup A))$ is isomorphic to $G(V-(S\cup B))$ under a vertex mapping that takes $R-M$ to $R-M$. Testing these conditions for all $M \subseteq R$ is prohibitively expensive, so stronger conditions must be imposed.

Let E_{ij} be the automorphism on $G(V-S)$ that swaps vertices i and j and maps each other vertex to itself. We define vertex symmetry function $sym(i,j)$ to be true iff E_{ij} is adjacency-preserving. Vertex symmetry has the following properties:

reflexive $sym(i,i)$ holds for all i since E_{ii} is the identity automorphism.

symmetric $sym(i,j)=sym(j,i)$ because $E_{ij}=E_{ji}$.

transitive $sym(i,j)$ and $sym(j,k)$ implies $sym(i,k)$ since compositions of adjacency-preserving E_{ij}'s are adjacency-preserving, and $E_{ik}=E_{ij}\circ E_{jk}\circ E_{ij}$.

Thus, vertex symmetry is an equivalence relation, so $sym(i,j)$ induces a partition of R. Let M be the largest vertex set in the partition. By the definition of $sym(i,j)$, M has the following properties:

1. the graph induced by M is either complete or empty
2. every $v \in V-(S\cup M)$ is either adjacent to every vertex in M or to no vertex in M.

Thus, $\forall\, A,B \subseteq M$, every isomorphism from $G(V-(S\cup A))$ to $G(V-(S\cup B))$ that maps $M-A$ to $M-B$ and maps each other vertex to itself is adjacency-preserving. Since these isomorphisms trivially map $R-M$ to $R-M$, $N(R-M,S\cup A)=N(R-M,S\cup A)$.

For an example of symmetry reduction for counting Hamiltonian Paths, see Figures 4 and 5.

3.2.3 S Subset Reduction

By the bounding theorem, in (12), $N(R-M,S\cup A_i)\geq N(R-M,S\cup A_{i+1})$. So, in computing the symmetry formula, if $N(R-M,S\cup A_i)=0$ for some i, then the remaining terms are all zero. Using this information saves $O((|M|-i)2^{|R-M|}poly(n))$ computation time.

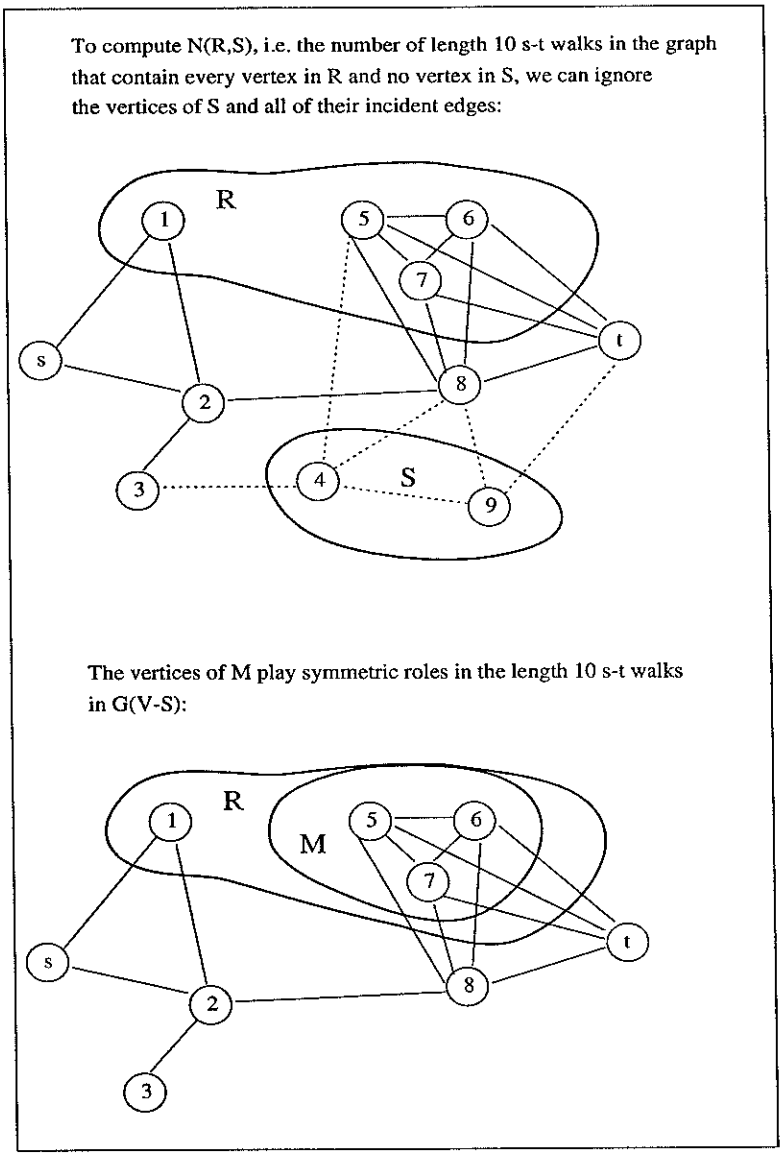
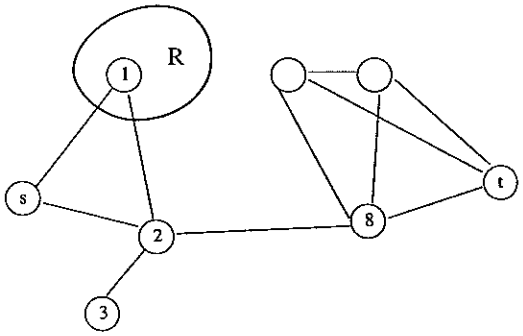
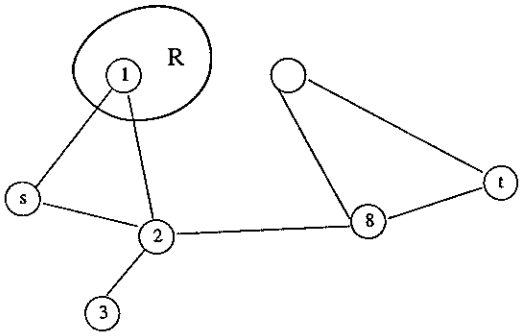


Figure 4: Identifying Symmetric $M \subseteq R$ When Counting Hamiltonian Paths

Moving any single element from M to S gives an isomorphic G(V-S).
N(R-M, S u {5}) = N(R-M, S u {6}) = N(R-M, S u {7}):



Moving any pair of elements from M to S gives an isomorphic G(V-S).
N(R-M, S u {5,6}) = N(R-M, S u {5,7}) = N(R-M, S u {6,7}):



Applying the recurrence N(R,S) = N(R-{r}, S) - N(R-{r}, S u {r})
successively for r=5, r=6, then r=7 gives the formula:
$N(R,S) = \Sigma_A (-1)^{|A|} N(R-\{5,6,7\}, S \cup A)$, where A is any subset of {5,6,7}.
By symmetry, this is:
N(R,S) = N(R-{5,6,7}, S) - 3 N(R-{5,6,7}, S u {5}) + 3 N(R-{5,6,7}, S u {5,6}) - N(R-{5,6,7}, S u {5,6,7}).

Figure 5: Using Symmetry to Count Hamiltonian Paths

3.3 Algorithm

Augmenting the recursion of Theorem 1 with the $|Q| = 1$ R subset reduction, the symmetry reduction, and the S subset reduction produces the algorithm:

```
N(R, S)
{
function N(∅, S);
integer i, value, sum = 0;
set A, M;

if (R = ∅) return(N(∅, S)); (* base case *)

for each r ∈ R
    if (N(∅, S) − N(∅, S ∪ {r}) = 0) return(0); (* R subset reduction *)

M=largest set in the partition of R induced by sym(); (* symmetry reduction *)

for (i = 0; i ≤ |M|; i++)
    A=first i elements of M;
    value=C(|M|, i) N(R − M, S ∪ A);
    if (value = 0) return(sum); (* S subset reduction *)
    sum = sum + value;

return(sum);
}
```

When $|M| = 1$, i.e. no symmetry is found, any $r \in R$ can be chosen as the "split" element in the recursion.

In the worst case, none of the reductions are used, so the computation is a depth first traversal of the binary tree induced by the recursion, with root $N(\{1 \ldots n\}, \emptyset)$, nodes $N(R, S)$, and leaves $N(\emptyset, S)$. Assuming that each $N(\emptyset, S)$ computation and each symmetry search takes $poly(n)$ time, the worst case time complexity is $O(2^n poly(n))$.

For the Hamiltonian path problem, partitioning R by $sym()$ takes $O(n|E|)$ time. Also, computing $N(S)$ requires $O(n|E|)$ time [8] for each $S \subseteq \{1 \ldots n\}$. Therefore, the algorithm to count Hamiltonian paths has time complexity $O(2^n n|E|)$.

Note that the first node to compute values of $N(\emptyset, S)$ and $N(\emptyset, S \cup \{r\})$ is the ancestor of all nodes that use these values. By keeping a stack of $N(\emptyset, S)$ values computed by ancestors of the current node, $N(\emptyset, S)$ is only computed once for each $S \subseteq \{1 \ldots n\}$. Since up to $n + 1$ $N(\emptyset, S)$ values may be computed in a node and the computation tree has depth n, the stack requires enough space

to store $O(n^2)$ $N(\emptyset, S)$ values. So the algorithm presented here maintains the polynomial space requirement of the earlier inclusion and exclusion algorithms.

4 Recurrence-Based Reduction Tests on the Hamiltonian Path Problem

4.1 Test

The Hamiltonian path algorithm described was tested on 100 random directed graphs $G_{n,p}$ (with $V = \{s,t\} \cup \{1 \ldots n\}$ and edge probability p) for each $n \in \{4, 6, \ldots, 18\}$ and $p \in \{0.00, 0.05, \ldots, 1.00\}$. The results for each n follow the pattern shown in Figure 6.

4.2 Test Results

The algorithm's efficiency varies with p. For very low p, $G_{n,p}$ is almost always disconnected, and in this case the R subset reduction finds that there are no s-t Hamiltonian paths in the first $N(R,S)$ call. As p increases, $G_{n,p}$ is more likely to be connected, and the S subset reduction begins to play a role. Then there is a p range in which $G_{n,p}$ transits from almost surely not having an s-t Hamiltonian path to almost surely having one. The effectiveness of the R and S subset reductions decrease during this transition. None of the reductions are very effective in the range of p for which $G_{n,p}$ almost certainly has an s-t Hamiltonian path, but very few of the $(n+1)!$ possible s-t Hamiltonian paths are present. Finally, for p near 1, $G_{n,p}$ is almost certainly very dense, and the symmetry reduction is highly effective.

4.3 Open Questions

There are several areas for further research related to the algorithms presented here. For the R subset reduction we use $|Q|$=1, but it is an open question whether the optimal value is some other constant, or depends on n, $|R|$, or $|S|$. Our search for symmetric vertices is efficient, but it only examines a few of the possible cases. Is there an efficient method to check for more cases of symmetry? Would less efficient methods be appropriate for highly symmetric graphs? Are real problems more likely to be symmetric than random test problems? Our reduction utilizes symmetry among vertices. In highly symmetric graphs it may be worthwhile to examine symmetry among subgraphs as well.

Pósa [14] shows that the minimum p value for which $G_{n,p}$ is almost surely Hamiltonian decreases to zero as n goes to infinity. Shamir [17] shows that the threshold p range in which $G_{n,p}$ is neither almost surely Hamiltonian nor almost surely non-Hamiltonian decreases to zero as n goes to infinity. These results are validated by Figure 11. The counting algorithm's loss of efficiency trails the increasing probability of $G_{n,p}$ having an s-t Hamiltonian path in Figure 6. For large n, is the algorithm efficient in the p threshold range? The algorithm's performance in the p threshold range affects its usefulness as a backup algorithm

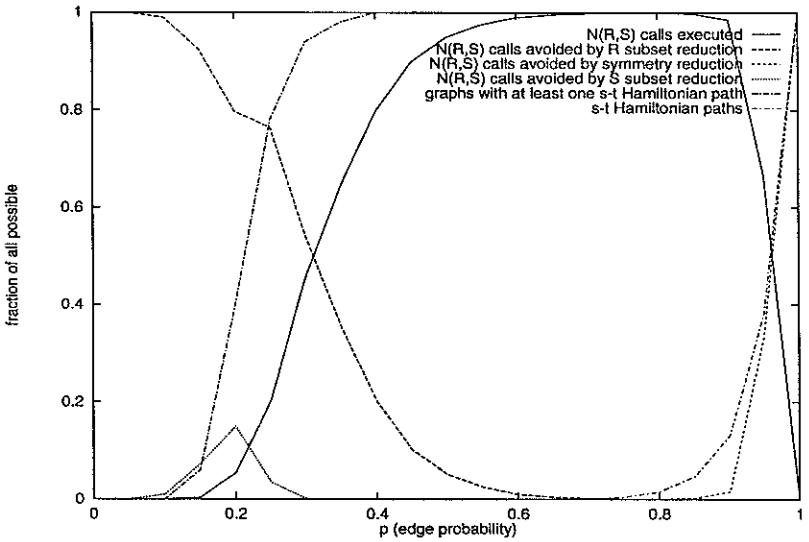


Figure 6: Test Results for n=18

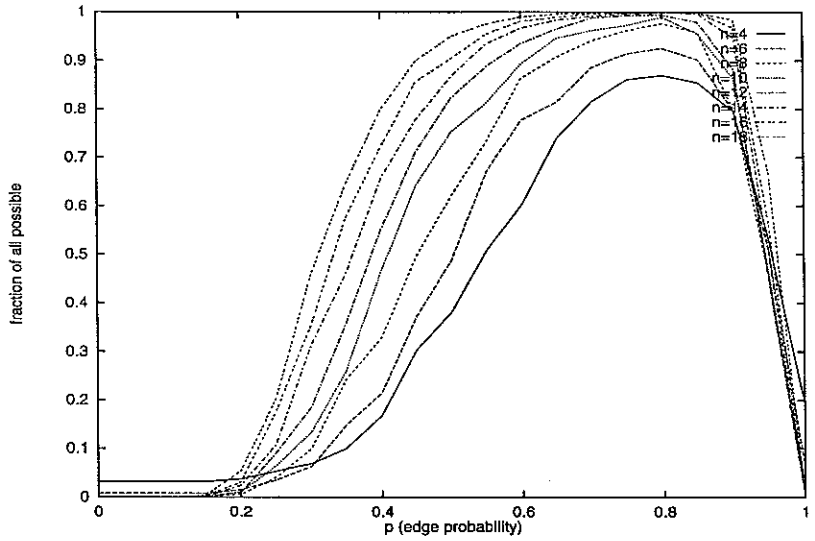


Figure 7: N(R,S) Calls Executed

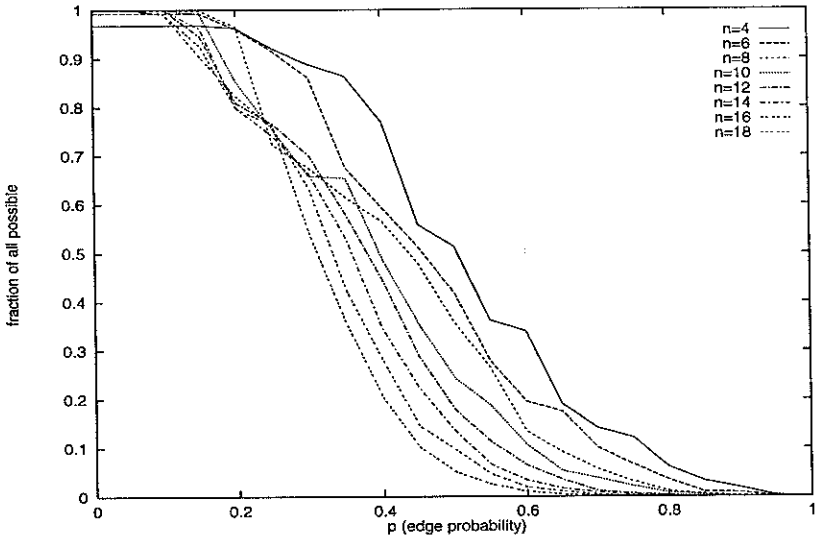


Figure 8: N(R,S) Calls Avoided by R Subset Reduction

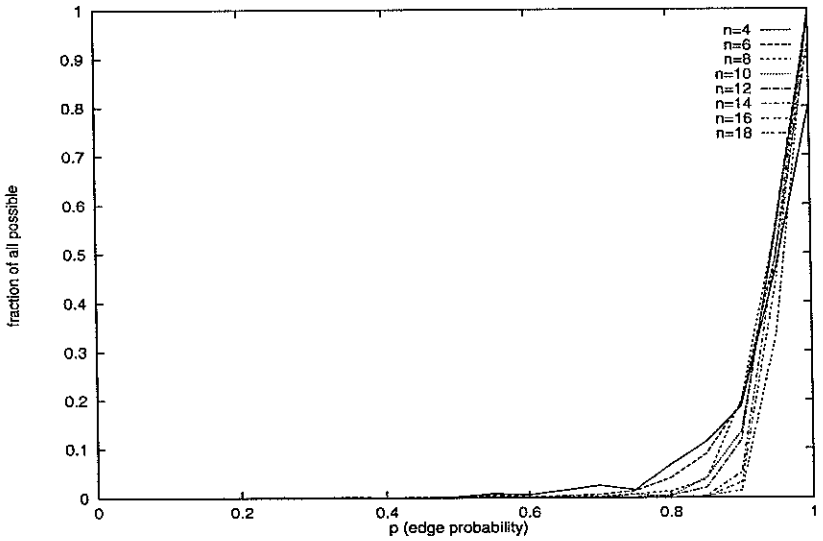


Figure 9: N(R,S) Calls Avoided by Symmetry Reduction

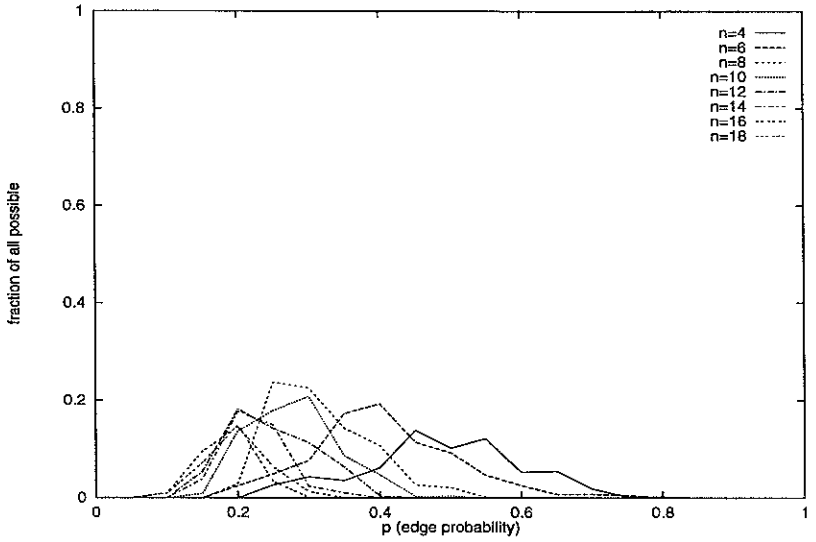


Figure 10: N(R,S) Calls Avoided by S Subset Reduction

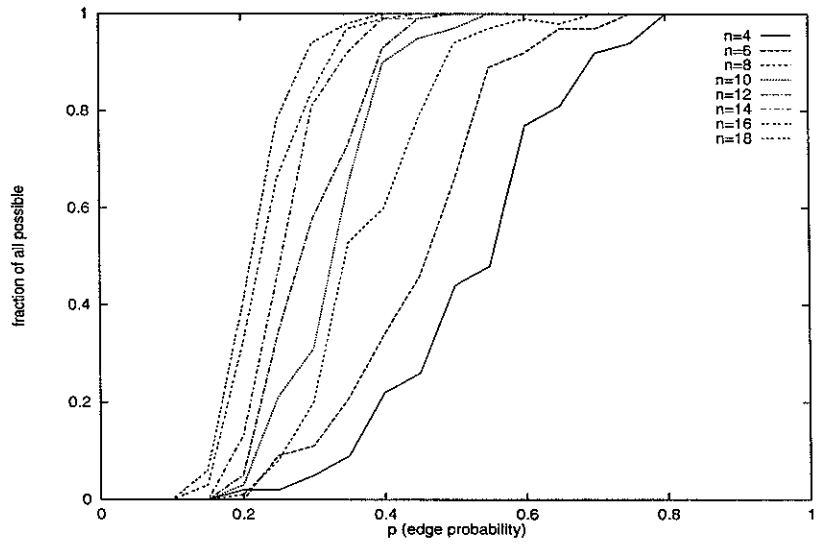


Figure 11: Graphs with at Least One s-t Hamiltonian Path

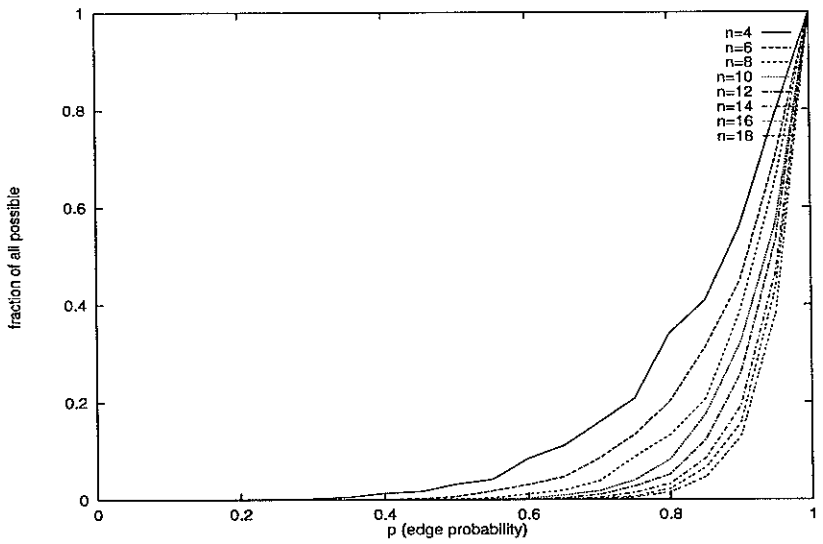


Figure 12: s-t Hamiltonian Paths

for probabilistic detection algorithms (e.g. [5]) that work well when $G_{n,p}$ is either almost surely Hamiltonian or almost surely non-Hamiltonian.

5 Tailoring the Algorithm to Other #P Problems

To use the algorithm on a given problem:

1. Phrase the problem as the size of the intersection of some sets $B_1, \ldots, B_n$.
2. Develop a method to compute $N(\emptyset, S)$ for each $S \subseteq \{1 \ldots n\}$.
3. Tailor the conditions of the symmetry theorem to structures in the problem.
4. Find a method to efficiently identify symmetric sets in problem instances.

The first two conditions are required to develop any inclusion and exclusion algorithm. The second two conditions are new challenges. Even when it is easy to define symmetric sets for a problem, it may be inefficient to search for all possible cases of symmetry. Instead, it is important to find fast searches that examine many of the most likely occurences of symmetry. It is not necessary to implement symmetry, i.e. the fourth condition can be trivially satisfied by always taking $M = \{r\}$ for some $r \in R$.

Clearly, the algorithm will have time complexity on the order of 2^n multiplied by the sum of the time required to compute $N(\emptyset, S)$ and the time taken to search for symmetry. Also, the algorithm will require enough space to store the problem, compute $N(S)$, and accumulate the sum of $N(S)$ over $S \subseteq \{1 \ldots n\}$. The algorithms presented here all have time complexity $O(2^n poly(n))$ and space complexity $O(poly(n))$.

To illustrate the process, we tailor the algorithm to the problems of counting vertex colorings, counting CNF satisfying assignments, and computing the permanent of a 0-1 matrix.

5.1 Vertex Coloring

The original inclusion and exclusion algorithm to count colorings is due to H. Whitney [21]. For a given k, an assignment is a labelling of the vertices in $G = (V, E)$ such that each vertex receives one of k colors. A monochrome edge has both incident vertices the same color. An edge that is incident to two vertices with different colors is polychrome. A vertex k-coloring is an assignment in which every edge is polychrome.

Let U be the set of all assignments. Label the edges so that $E = \{1 \ldots n\}$. Let B_i be the set of assignments in which edge i is polychrome. Then $|B_1 \cap \ldots \cap B_n|$ is the number of k-colorings of G.

By definition, $N(\emptyset, S)$ is the number of assignments in which every edge in S is monochrome. Note that if a pair of vertices is connected by a path of monochrome edges, then the vertices must have the same color. Let $G(S)$ be the

graph induced by edge subset S. Then, in the assignments counted by $N(\emptyset, S)$, each connected component of $G(S)$ has all vertices the same color. Since there are k possible colors for each connected component:

$$N(\emptyset, S) = k^{cc(G(S))} \tag{13}$$

where $cc(G(S))$ is the number of connected components in $G(S)$.

Hence, the number of vertex k-colorings of graph G is:

$$\sum_{S \subseteq \{1 \ldots n\}} (-1)^{|S|} k^{cc(G(S))} \tag{14}$$

Examples to illustrate terms and calculations for counting vertex colorings are shown in Figures 13 and 14.

The symmetry requirements are similar to those for the Hamiltonian path algorithm. To use the symmetry reduction, make M the edges of the largest clique such that:

1. All edges in the clique are in R
2. If there is an edge in R or S connecting any clique vertex to a vertex v, then there are edges in R or S, respectively, connecting each clique vertex to v.

Let $\hat{E}_{ij}$ be the automorphism on G that swaps vertices i and j and maps each other vertex to itself. Define $\hat{sym}(i,j)$ to be true iff $\hat{E}_{ij}$ preserves adjacencies by edges in R and also preserves adjacencies by edges in S. Similar to the vertex symmetry function for the Hamiltonian path problem, $\hat{sym}(i,j)$ is an equivalence relation.

Each set of vertices in the partition of V induced by $\hat{sym}(i,j)$ has exactly one of the following properties:

1. The vertices form a clique of R edges.
2. The vertices form a clique of S edges.
3. The set has no pair of adjacent vertices.

The edges of the R cliques satisfy our conditions for the symmetric set M. So choose M to be the edges of the largest R clique among the vertex sets in the partition of V induced by $\hat{sym}(i,j)$.

Figures 15 and 16 illustrate symmetry in vertex coloring.

5.2 CNF Satisfying Assignments

To count satisfying assignments for a CNF formula with n clauses, let U be the set of all assignments, and let B_i be the set of assignments that satisfy the ith

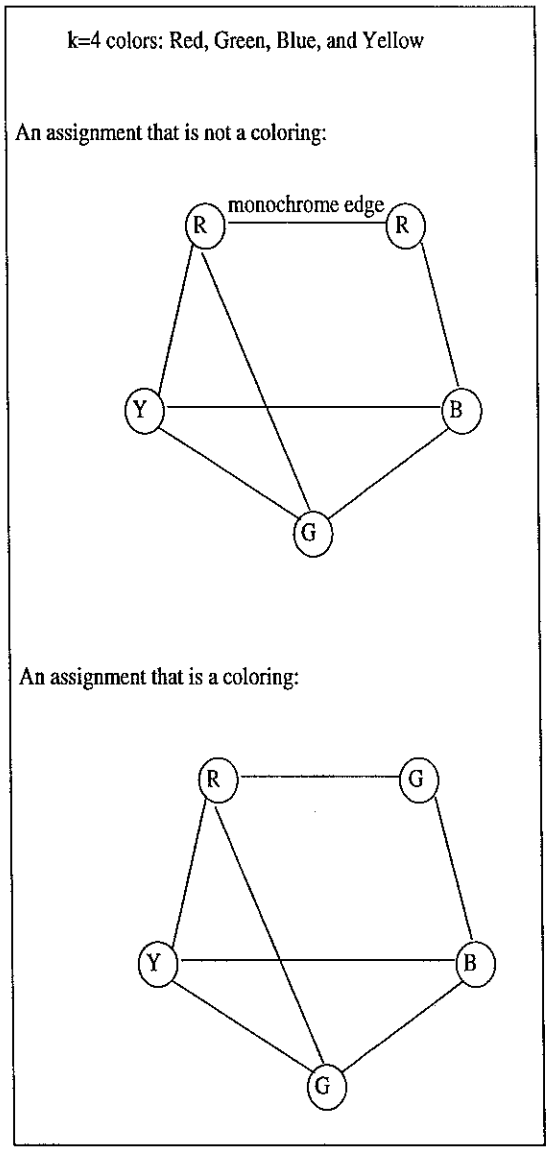
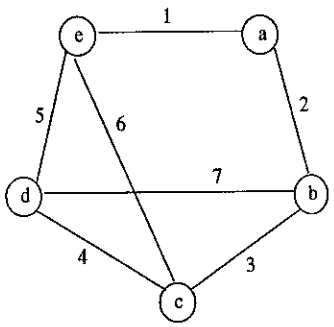


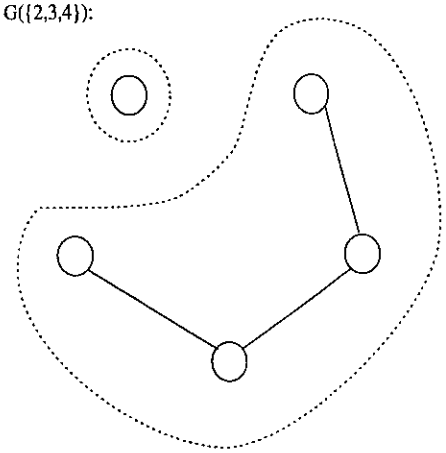
Figure 13: Assignment and Coloring

G with numbered edges and labelled vertices:



N({},{2,3,4}) is the number of assignments in which edges 2, 3, and 4 are monochrome. In such assignments, vertices a, b, c, and d share a single color.

G({2,3,4}):



Each connected component of G({2,3,4}) receives one of the k colors, so $N(\{\},\{2,3,4\})=k^2$.

Figure 14: Example Computation of $N(\emptyset, S)$ for Coloring

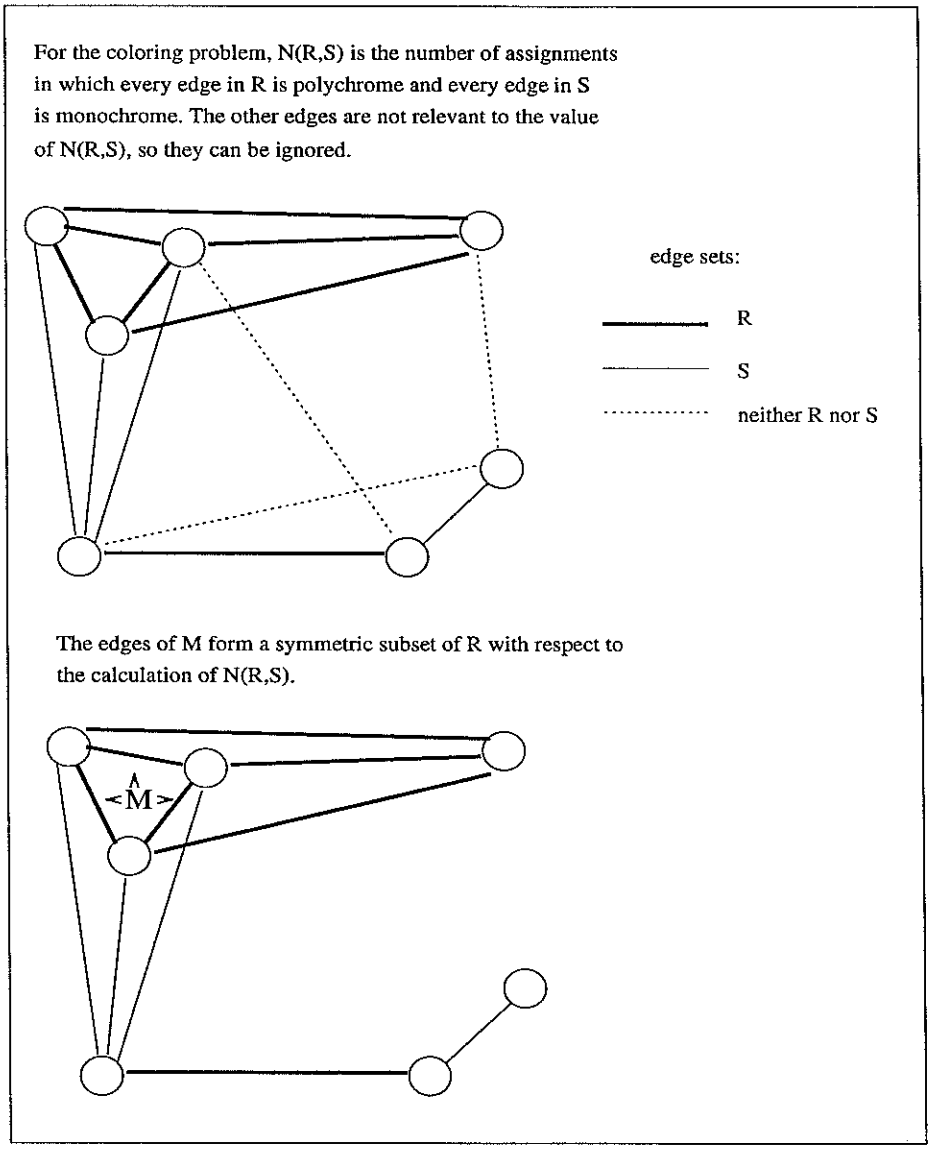


Figure 15: Identifying Symmetric $M \subseteq R$ to Count Colorings

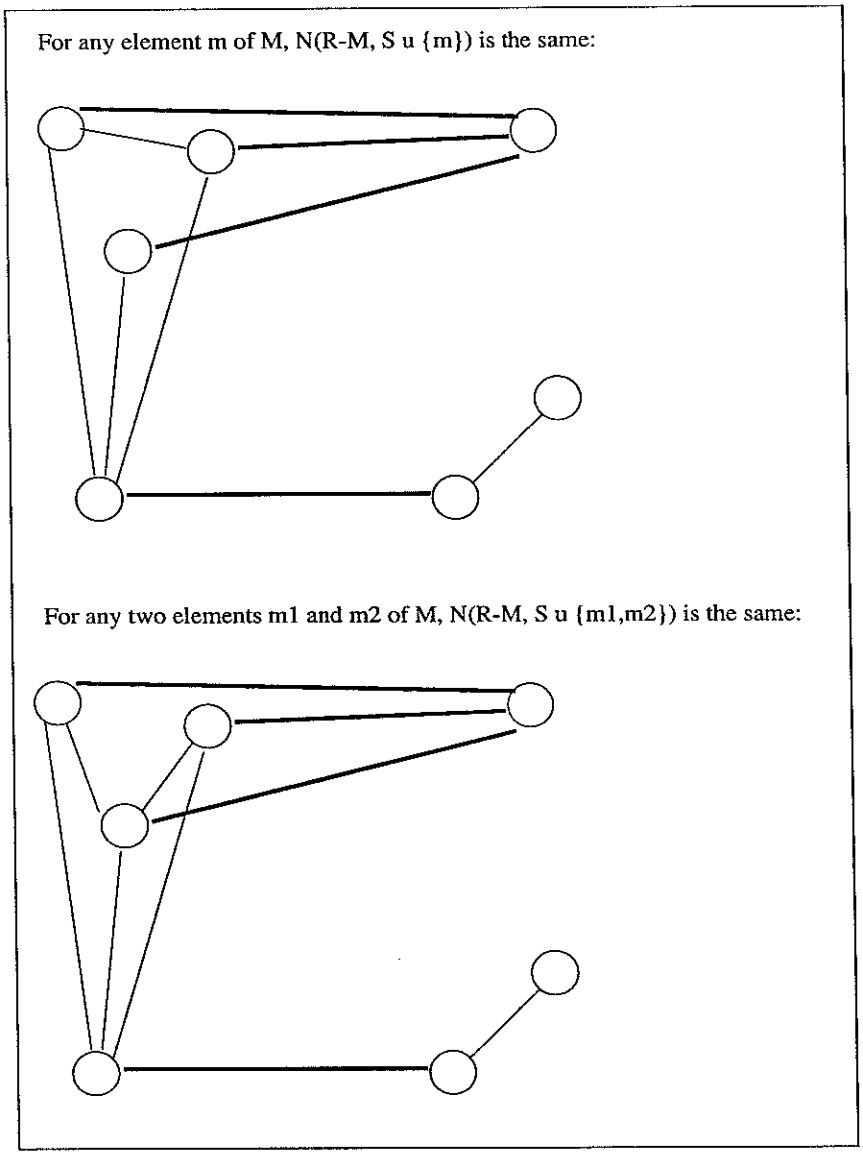


Figure 16: Using Symmetry to Count Colorings

clause. Then $|B_1 \cap \ldots \cap B_n|$ is the number of assignments that satisfy the entire formula.

By definition, $N(\emptyset, S)$ is the number of assignments that do not satisfy any clause indexed by S. $N(\emptyset, S)$ is zero if both a variable and its negation occur in the clauses indexed by S. Otherwise, $N(\emptyset, S)$ is $2^{m(S)}$, where $m(S)$ is the number of variables that do not occur in any clause indexed by S. This algorithm was developed by N. Linial and N. Nisan [10].

An example will clarify the terms and calculations. Consider the CNF formula:

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\overline{x_3} \vee x_4) \tag{15}$$

The assignment $x_1 = T$, $x_2 = F$, $x_3 = F$, $x_4 = F$ satisfies the formula:

$$(T \vee F) \wedge (T \vee F) \wedge (\overline{F} \vee F) \tag{16}$$

$$= (T \vee F) \wedge (T \vee F) \wedge (T \vee F) \tag{17}$$

$$= T \wedge T \wedge T \tag{18}$$

To satisfy the formula, an assignment must satisfy all 3 clauses. The assignment $x_1 = T$, $x_2 = F$, $x_3 = T$, $x_4 = F$ does not satisfy the formula because the third clause is not satisfied.

By definition, $N(\emptyset, \{1,2\})$ is the number of assignments that satisfy neither the first nor the second clause. Thus, $N(\emptyset, \{1,2\})$ is the number of assignments that satisfy:

$$\overline{(x_1 \vee x_2)} \wedge \overline{(x_1 \vee x_3)} \tag{19}$$

{DeMorgan's Law}

$$(\overline{x_1} \wedge \overline{x_2}) \wedge (\overline{x_1} \wedge \overline{x_3}) \tag{20}$$

$$\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3} \tag{21}$$

The values of x_1, x_2, and x_3 are determined – they must all be F. However, x_4, which does not occur in the first or second clauses, may take on either value. So $N(\emptyset, \{1,2\})$ is 2^1.

As an example of the case $N(\emptyset, S) = 0$, $N(\emptyset, \{2,3\})$ is the number of assignments that satisfy:

$$\overline{(x_1 \vee x_3)} \wedge \overline{(\overline{x_3} \vee x_4)} \tag{22}$$

{DeMorgan's Law}

$$= (\overline{x_1} \wedge \overline{x_3}) \wedge (\overline{\overline{x_3}} \wedge \overline{x_4}) \tag{23}$$

$\{\overline{\overline{x}} \equiv x\}$

$$= (\overline{x_1} \wedge \overline{x_3}) \wedge (x_3 \wedge \overline{x_4}) \tag{24}$$

$$= \overline{x_1} \wedge \overline{x_3} \wedge x_3 \wedge \overline{x_4} = F \tag{25}$$

No assignment satisfies a false formula, so $N(\emptyset, \{2,3\}) = 0$.

To develop intuition about symmetry in counting CNF assignments, consider the formula:

$$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6) \tag{26}$$

$N(\{3,4,5\},\{1,2\})$ is the number of assignments that satisfy neither of the first two clauses and satisfy all of the remaining clauses. So $N(\{3,4,5\},\{1,2\})$ is the number of satisfying assignments of the formula:

$$\overline{(x_1 \vee x_2)} \wedge \overline{(x_1 \vee \overline{x_3})} \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6) \tag{27}$$

{DeMorgan's Law}

$$= (\overline{x_1} \wedge \overline{x_2}) \wedge (\overline{x_1} \wedge \overline{\overline{x_3}}) \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6) \tag{28}$$

$$= \overline{x_1} \wedge \overline{x_2} \wedge x_3 \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6) \tag{29}$$

Note that x_1, x_2, and $\overline{x_3}$, i.e. all of the terms in the clauses of S, must be F for this formula to be true. But then occurences of these S terms in the clauses of R are irrelevant:

$$= (x_1 = F) \wedge (x_2 = F) \wedge (\overline{x_3} = F) \wedge (F \vee x_4 \vee x_5) \wedge (F \vee x_4 \vee x_5) \wedge (F \vee x_4 \vee x_6) \tag{30}$$

$$= (x_1 = F) \wedge (x_2 = F) \wedge (\overline{x_3} = F) \wedge (x_4 \vee x_5) \wedge (x_4 \vee x_5) \wedge (x_4 \vee x_6) \tag{31}$$

Since the clauses resulting from the third and fourth clauses in the original formula are equal in the $N(R,S)$ formula, these clauses are symmetric with respect to counting the satisfying assignments of $N(R,S)$. Thus, $M = \{3,4\}$.

If a term occurs in some clause indexed by S, then its occurences in clauses indexed by R do not affect the value of $N(R,S)$. So for each clause indexed by R, remove all variables that occur in the same state of negation in some clause indexed by S. Then make M the largest set of equal clauses indexed by R.

Requiring one or more of the equal clauses to be satisfied is equivalent to requiring only one of them to be satisfied, so all but one of the equal clauses can be eliminated from R, i.e. $N(R,S) = N((R - M) \cup \{a\}, S)$ where $a \in M$. Instead of using the symmetry formula 12, simply eliminate all but one of the elements of M from R.

5.3 Permanent of a 0-1 Matrix

The permanent of an $n \times n$ matrix A is:

$$per(A) \equiv \sum_{j_1 \ldots j_n} a_{1j_1} \ldots a_{nj_n} \tag{32}$$

where $j_1 \ldots j_n$ is a permutation of $1 \ldots n$. Hence, the permanent is the sum of all product terms with exactly one element from each row and exactly one element from each column.

Define a row term to be a set of n nonzero elements with exactly one element from each row. The number of row terms in matrix A is the product of the row sums:

$$\prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \tag{33}$$

The permanent is the number of sets of n elements with exactly one element from each row and exactly one element from each column. Hence, the permanent is the number of row terms that have exactly one element from each column.

Let U be the set of row terms. Define B_i to be the set of row terms that contain at least one element from column i. By the pigeonhole principle, the permanent is $|B_1 \ldots B_n|$.

By definition, $N(\emptyset, S)$ is the number of row terms that do not contain elements from any of the columns indexed by S. Hence, $N(\emptyset, S)$ can be computed by zeroing the columns indexed by S and taking the product of row sums:

$$N(\emptyset, S) = \prod_{i=1}^{n} \sum_{j \notin S} a_{ij} \tag{34}$$

Substituting into the formula (7) gives:

$$perA = \sum_{S \subseteq \{1 \ldots n\}} (-1)^{|S|} (\prod_{i=1}^{n} \sum_{j \notin S} a_{ij}) \tag{35}$$

This algorithm is the work of H. J. Ryser [16].

Since the product of row sums is invariant under permutations of the columns, to use the symmetry reduction, make M the largest set of equal columns in R.

6 Zero Sets and Vestigial Elements

When the search for symmetry fails, the algorithm is free to choose the split element $r \in R$ for the recursion $N(R,S) = N(R-\{r\},S) - N(R-\{r\},S\cup\{r\})$. Some split elements lead to greater reductions in computation than others. Knowledge of the interaction between the recursion and the structure of the problem instance can guide the selection of effective split elements.

6.1 Zero Sets

Consider the graph in Figure 17. Note that every s-t walk in G contains vertex 4, vertex 5, or both. Since $N(R,S)$ counts s-t walks which lack all vertices in S, if $\{4,5\} \subseteq S$ then $N(R,S) = 0$.

If R contains vertices 4 and 5, then choosing vertices 4 and 5 as the split elements in the next two recursions yields a reduction as follows:

$$N(R,S) = N(R-\{4\},S) - N(R-\{4\},S\cup\{4\}) \tag{36}$$

$$= N(R-\{4,5\},S) - N(R-\{4,5\},S\cup\{5\}) - N(R-\{4,5\},S\cup\{4\}) + N(R-\{4,5\},S\cup\{4,5\}) \tag{37}$$

$N(R-\{4,5\},S\cup\{4,5\}) = 0$ since $\{4,5\} \subseteq S$. So

$$N(R,S) = N(R-\{4,5\},S) - N(R-\{4,5\},S\cup\{5\}) - N(R-\{4,5\},S\cup\{4\}) \tag{38}$$

and $\frac{1}{4}$ of the computation of $N(R,S)$ can be avoided. This example motivates the general notion of zero sets. (In the Hamiltonian path algorithm, every vertex cut set that separates s from t is a zero set.)

Definition 2 (Zero Set) *A zero set is a set $Z \subseteq \{1 \ldots n\}$ such that $N(\emptyset,S) = 0$ $\forall S \supseteq Z$.*

Theorem 5 (Zero Set Reduction) *If $Z \subseteq S$ is a zero set then $N(R,S) = 0$.*

Proof. All terms in $N(R,S) = \sum_{A\subseteq R}(-1)^{|A|}N(\emptyset,S\cup A)$ (6) have $Z \subseteq S\cup A$. By the definition of a zero set, every term is zero. □

6.2 Vestigial Elements

Now consider the graph in Figure (18). Suppose we wish to compute $N(\{1,4,5,6\},\{2,3\})$ – the number of length 7 s-t walks that contain vertices 1, 4, 5, and 6, and that lack vertices 2 and 3. Since $\{2,3\}$ is a cut set that isolates vertex 4 from s and t, no s-t walk that lacks vertices 2 and 3 can contain vertex 4. So $N(\{1,4,5,6\},\{2,3\}) = 0$. In general, if S contains a vertex cut set that isolates s or t from some vertex in R, then $N(R,S) = 0$. We call the isolated vertex a vestigial element, and we call its isolating cut set its structure.

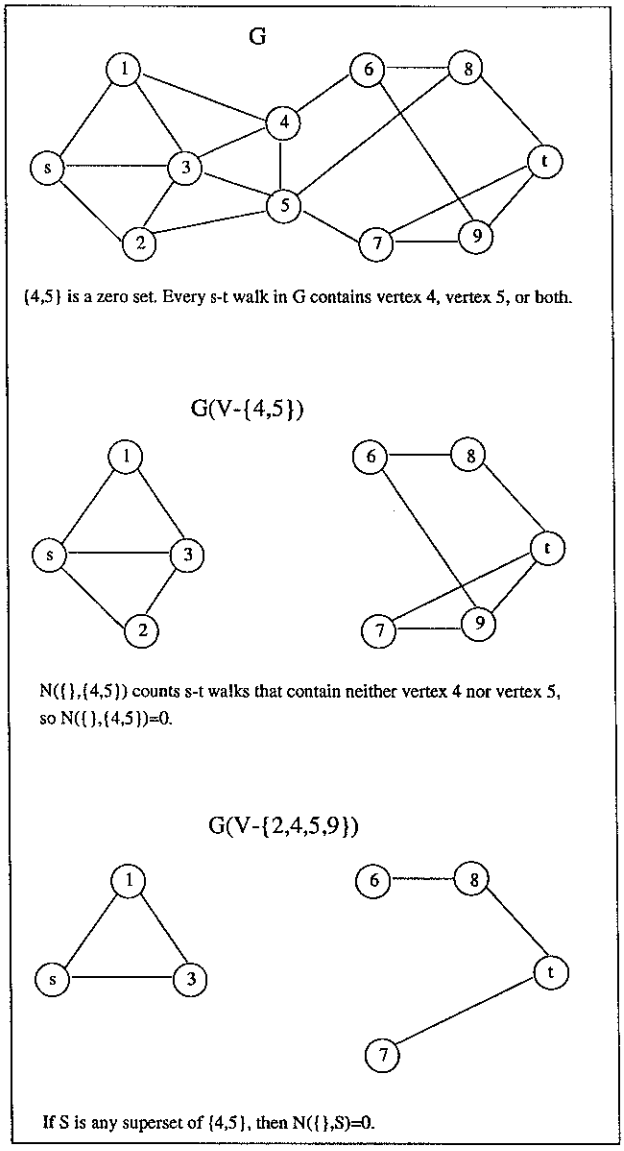


Figure 17: A Zero Set in the Algorithm to Count Hamiltonian Paths

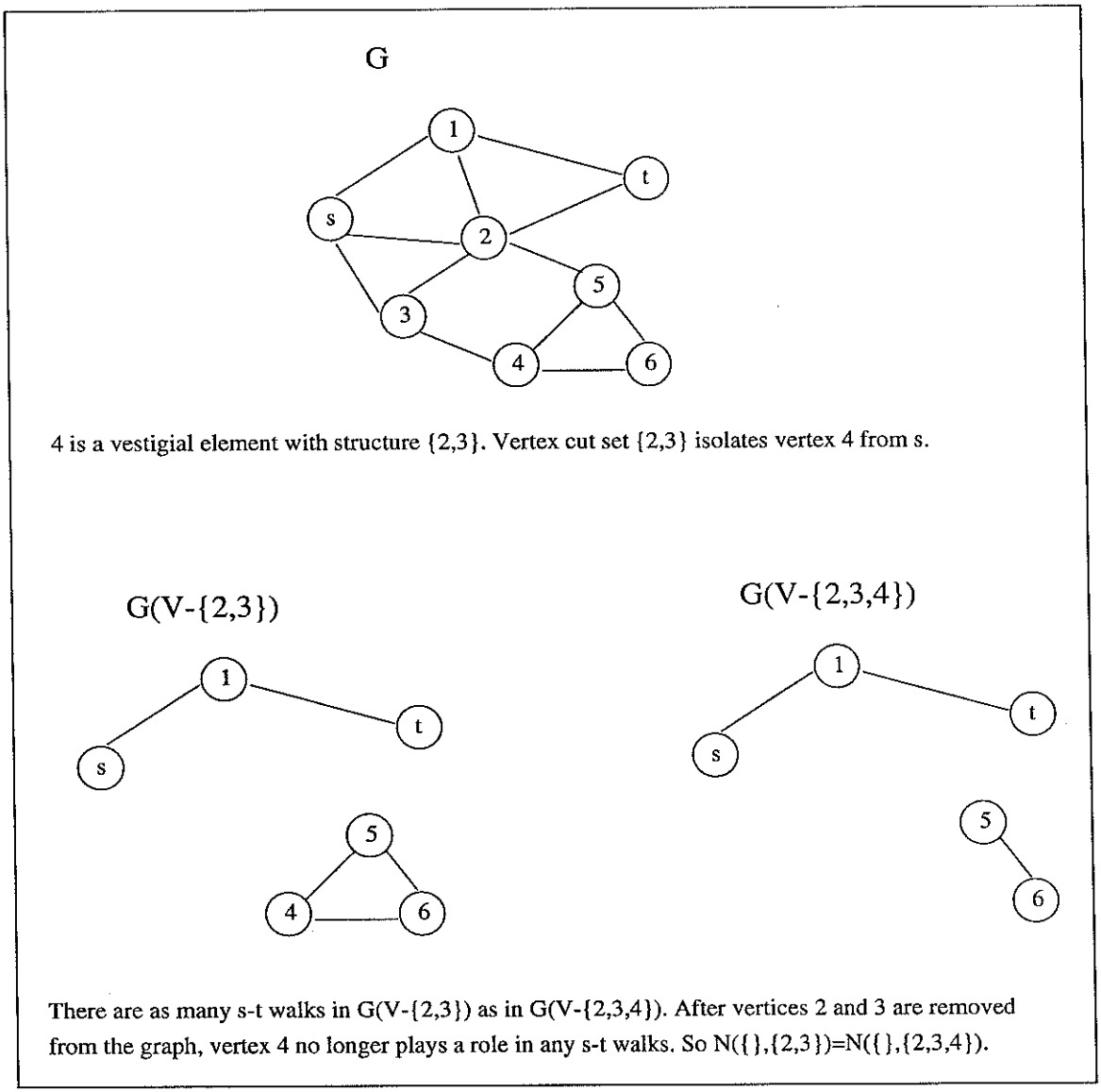


Figure 18: A Vestigial Element in the Algorithm to Count Hamiltonian Paths

Definition 3 (Vestigial Element) *$v \in \{1 \ldots n\}$ is a vestigial element with structure $X \subseteq \{1 \ldots n\} - \{v\}$ if $N(\emptyset, S) = N(\emptyset, S \cup \{v\})\ \forall S \supseteq X$.*

Theorem 6 (Vestigial Element Reduction) *If v is a vestigial element with structure X, $v \in R$, and $X \subseteq S$, then $N(R, S) = 0$.*

Proof. By the bounding theorem, $\forall\ Q \subseteq R$ and $T \subseteq S$, $N(Q, T) \geq N(R, S)$. Setting $Q = \{v\}$ and $T = S$ gives:

$$N(R, S) \leq N(\{v\}, S) \tag{39}$$

Apply the recursion to the RHS:

$$N(R, S) \leq N(\emptyset, S) - N(\emptyset, S \cup \{v\}) \tag{40}$$

By the definition of a vestigial element, the RHS is zero. □

The next two theorems do not apply directly to the development of $N(R, S)$ algorithms. However, they relate the notion of vestigial elements to interesting work by other researchers [9, 11, 21].

Theorem 7 (Vestigial Element Cancellation) *If v is a vestigial element with structure X, then:*

$$\sum_{S \supseteq X} (-1)^{|S|} N(\emptyset, S) = 0 \tag{41}$$

Proof. Each $S \supseteq X$ with $v \notin S$ can be paired with $S \cup \{v\} \supseteq X$. These pairs cancel in (41) because $(-1)^{|S|}$ and $(-1)^{|S \cup \{v\}|}$ have opposite signs. (This cancellation theorem is the special case of the reduction theorem in which $R = \{1 \ldots n\} - S$.) □

Theorem 8 (Vestigial Element Interference) *If $v_1 \ldots v_K$ are vestigial elements with structures $X_1 \ldots X_K$, and $\forall j < k$, $v_k \notin X_j$, then:*

$$\sum_{(S \supseteq X_1) \vee \ldots \vee (S \supseteq X_K)} (-1)^{|S|} N(\emptyset, S) = 0 \tag{42}$$

and hence:

$$|B_1 \cap \ldots \cap B_n| = \sum_{(S \not\supseteq X_1) \wedge \ldots \wedge (S \not\supseteq X_K)} (-1)^{|S|} N(\emptyset, S) \tag{43}$$

Proof. Consider a mapping on sets S such that $S \supseteq X_1 \vee \ldots \vee S \supseteq X_K$: Let k be the minimum index for which $S \supseteq X_k$. If $v_k \in S$, $S \rightarrow S - \{v_k\}$, otherwise $S \rightarrow S \cup \{v_k\}$, i.e. toggle v_k.

Since $\forall j < k$, $v_k \notin X_j$, the mapping is its own inverse. So the mapping partitions the sets into pairs of the form $(S, S \cup \{v_k\})$, where $S \supseteq X_k$. Each pair's terms cancel in the sum (42). □

6.3 Completers

Definition 4 (Completer) *If v is a vestigial element with structure X; $c, v \in R$; and $X \subseteq S \cup \{c\}$, then c is a completer. If Z is a zero set, $c \in R$, and $Z \subseteq S \cup \{c\}$, then c is a completer.*

The completer reduction unifies zero set and vestigial element reductions:

Theorem 9 (Completer Reduction) *If c is a completer then $N(R,S) = N(R - \{c\}, S)$.*

Proof. Apply the recursion (3) with $r = c$:

$$N(R,S) = N(R - \{c\}, S) - N(R - \{c\}, S \cup \{c\}) \tag{44}$$

By the definition of a zero set or a vestigial element, $N(R - \{c\}, S \cup \{c\}) = 0$. □

6.4 Using Zero Sets, Vestigial Elements, and Completers

If there is a zero set $Z \subseteq R \cup S$, then choosing the members of $Z \cap R$ as succesive split elements achieves a reduction after $|Z \cap R|$ recursions. If there is a vestigial element $v \in R$ with structure $X \subseteq R \cup S$, then choosing the members of $X \cap R$ as succesive split elements achieves a reduction after $|X \cap R|$ recursions.

Ideally, at every level of recursion the algorithm would determine a smallest $Q \subseteq R$ such that $N(R-Q, S \cup Q) = 0$. Then some element of Q would be chosen as the split element. In practice, determining such a set Q can be prohibitively expensive, so heuristics are required.

If there is an immediate zero set or vestigial element reduction, then $N(\emptyset, S) - N(\emptyset, S \cup \{r\}) = 0$ for some $r \in R$. To search for these reductions, the algorithm calculates $N(\emptyset, S \cup \{r\})$ $\forall r \in R$. However, for some problems it is more efficient to search directly for the vestigial element or zero set required to apply a reduction. For example, to search for an immediate zero set reduction in the algorithm to count Hamiltonian paths, it is faster to search for a cut vertex in R that separates s from t than to calculate $N(\emptyset, S \cup \{r\})$ $\forall r \in R$ by counting s-t walks [8]. So if there is a high probability of cut vertices, it is efficient to search for them directly.

Note that if the zero set, vestigial element, and completer reductions do not apply to $N(R,S)$, then they do not apply to the left child $N(R - \{r\}, S)$, but they may apply to the right child $N(R - \{r\}, S \cup \{r\})$. If the zero set, vestigial element, and completer reductions are applied to the initial node $N(\{1 \ldots n\}, \emptyset)$, and the completer reduction is always applied to the right child – making the recursive call $N(R - \{r, c_1, \ldots, c_m\}, S \cup \{r\})$ where $c_1 \ldots c_m$ are completers – then the zero set and vestigial element reductions are never used after the initial node. The completer reduction prunes all recursive calls that would result

in vestigial element or zero set reductions. This forward-pruning strategy is very useful when recursive calls have a high cost, e.g. when the algorithm's recursive calls are implemented as a diffusing computation on a message-passing multicomputer.

6.5 Zero Sets and Vestigial Elements in Specific Problems

6.5.1 Hamiltonian Paths

If a vertex cut set Z separates s from t, then Z is a zero set.

If X is a vertex cut set that separates s from some $v \in \{1 \ldots n\}$, then v is a vestigial element with structure X.

To apply the zero set and vestigial element reductions, find the connected component of $G(V - S)$ that contains s. If the component does not contain all vertices in $R \cup \{t\}$, then $N(R, S) = 0$. To apply the completer reduction, remove from R all cut vertices and all vertices that are not in the same connected component as s.

For each vertex cut set X that separates s from some vertex in $\{1 \ldots n\}$, let v be the highest-numbered vertex that is separated from s in $G(X)$. If $v < x$ for some $x \in X$, then throw out cut set X. Arrange the remaining cut sets $X_1 \ldots X_K$ in ascending order of v's. There is no interference among cancellations by the vestigial elements with cut sets $X_1 \ldots X_K$.

6.5.2 Vertex Colorings

Define $G(S)$ to be the subgraph induced by edge set S. Recall that $N(\emptyset, S)$ is determined by the number of connected components in $G(S)$. If $Q \subseteq E$ is a cycle, let v be an edge in Q, and let $X = Q - \{v\}$. If $S \supseteq X$, then adding or removing v from S does not change the number of connected components in $G(S)$. So v is vestigial with structure X.

If we order all cycles $C_1 \ldots C_K$ in ascending order of highest-numbered edges, then the conditions of vestigial element interference theorem are satisfied by assigning the highest-numbered edge in C_i to v_i and setting $X_i = C_i - v_i$. (The X_i are Whitney's broken cycles [21].)

$N(R, S)$ counts the assignments in which the vertices of each edge in R have separate colors, and every component of $G(S)$ is monochrome. By the vestigial element reduction, if there is an edge in R that connects vertices in the same component of $G(S)$, then $N(R, S) = 0$. By the completer reduction, if each edge in a set $\{v, c_1, \ldots, c_m\} \subseteq R$ connects the same pair of $G(S)$ components, then edges $c_1 \ldots c_m$ are completers, and hence they are redundant.

The $N(R, S)$ recursion, augmented with completer reductions, leads logically to a recursion which reduces a graph coloring problem instance to a pair of simpler instances of the same problem. Let $H(R, S)$ be the graph with one vertex for each connected component in $G(S)$, and an edge between vertices

if the corresponding $G(S)$ components are connected by an edge in R. Then $N(R,S)$ counts the k-colorings of $H(R,S)$. By the vestigial element reduction, if $H(R,S)$ has a self-loop, then $N(R,S) = 0$. By the completer reduction, if $H(R,S)$ has multiple edges between a pair of vertices, they can be reduced to a single edge.

To form $H(R-\{r\},S)$, remove edge r from $H(R,S)$; to form $H(R-\{r\},S\cup\{r\})$, contract edge r – combine its vertices to make a single vertex with the union of the original vertices' adjacencies. This is the contraction-deletion formula for graph coloring [15].

The computation is reduced by multiple edges being combined when we take the union of adjacencies. (This is the completer reduction at work.) So choose the split edge r to be the edge shared by the connected vertex pair with the most adjacencies in common.

There are many vertex coloring results that could contribute further to the algorithm. For example, trees have $k(k-1)^{|V|}$ k-colorings, where $|V|$ is the number of vertices in the tree. (Color each parent before its children. There are k choices for the root, $k-1$ thereafter.) So if $H(R,S)$ is a tree there is no need to continue the recursion [12]. Also, if $H(R,S)$ is disconnected, then the number of k-colorings is the product of the numbers of k-colorings of its components. So the component colorings can be computed separately. For further discussion, references, and results on coloring, see [15].

6.5.3 CNF Satisfying Assignments

Recall that $N(\emptyset,S)$ is zero if both a variable and its negation occur in the clauses indexed by S. Otherwise, $N(\emptyset,S)$ is $2^{m(S)}$, where $m(S)$ is the number of variables that do not occur in any clause indexed by S.

If Z is a pair of clauses containing a variable and its negation, then Z is a zero set.

Clause v is vestigial with structure X if each variable in v occurs in the same state of negation in some clause in X. (If $S \supseteq X$, then adding v to S does not change $N(\emptyset,S)$.)

To apply the completer reduction for zero sets, remove from R any clause containing a term for which the negation is in some clause of S. To apply the completer reduction for vestigial elements, remove from R any clause with each of its variables in some clause of S.

Consider only those vestigial element and structure pairs in which v follows every clause in X. Order $X_1 \ldots X_K$ so that $v_1 \leq \ldots \leq v_K$, and the cancellations associated with vestigial elements $v_1 \leq \ldots \leq v_K$ are non-interfering.

Combining the symmetry and completer reductions with the $N(R,S)$ recursion leads logically to a recursion which counts satisfiers for a CNF expression by counting the satisfiers for a pair of simpler CNF expressions.

Define $K(C,f)$ to be the number of assignments that satisfy CNF C, given that there are f free variables – variables that do not occur in any of the clauses

of C. In general, $K(C, f)$ is 2^f times the number of assignments that satisfy C. When C contains no clauses, define $K(\mathrm{C}, f)$ to be 2^f.

Consider the CNF instance:

$$C = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_5 \vee x_6) \wedge (x_1 \vee x_5 \vee x_6) \quad (45)$$

The number of satisfying assignments is $N(\{1,2,3,4,5\}, \emptyset) = K(C, 0)$. Using clause 1 as the split element:

$$N(\{1,2,3,4,5\}, \emptyset) = N(\{2,3,4,5\}, \emptyset) - N(\{2,3,4,5\}, \{1\}) \quad (46)$$

$N(\{2,3,4,5\}, \emptyset)$ is the number of assignments to variables x_1 through x_6 that satisfy:

$$C' = (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_5 \vee x_6) \wedge (x_1 \vee x_5 \vee x_6) \quad (47)$$

Since x_3 does not occur in C', every assignment to the other variables that satisfies C' still satisfies C' whether x_3 is T or F. So x_3 is the one free variable, and:

$$N(\{2,3,4,5\}, \emptyset) = K(C', 1) \quad (48)$$

$N(\{2,3,4,5\}, \{1\})$ is the number of assignments to variables x_1 through x_6 that satisfy:

$$\overline{(x_1 \vee x_2 \vee x_3)} \wedge (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_5 \vee x_6) \wedge (x_1 \vee x_5 \vee x_6) \quad (49)$$

By DeMorgan's law:

$$= \overline{(x_1} \wedge \overline{x_2} \wedge \overline{x_3} \wedge (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_5 \vee x_6) \wedge (x_1 \vee x_5 \vee x_6) \quad (50)$$

All satisfying assignments have $x_1 = x_2 = x_3 = F$. So fix their values:

$$= (T \vee x_4) \wedge (F \vee x_5) \wedge (F \vee x_5 \vee x_6) \wedge (F \vee x_5 \vee x_6) \quad (51)$$

Since $(\overline{x_1} \vee x_4)$ contains x_1 in the opposite state of negation from the split clause $(x_1 \vee x_2 \vee x_3)$, the term is fixed to be T, and thus the whole clause is always T regardless of the value assigned to x_4, so the clause can be eliminated from the CNF. (This is a completer reduction). Since x_4 can assume either value, and it no longer occurs in the CNF, x_4 becomes a free variable.

$$= (F \vee x_5) \wedge (F \vee x_5 \vee x_6) \wedge (F \vee x_5 \vee x_6) \quad (52)$$

False terms can be eliminated from the clauses:

$$= (x_5) \wedge (x_5 \vee x_6) \wedge (x_5 \vee x_6) \tag{53}$$

By the symmetry reduction, duplicated clauses can be merged:

$$C'' = (x_5) \wedge (x_5 \vee x_6) \tag{54}$$

Thus:

$$N(\{2,3,4,5\},\{1\}) = K(C'',1) \tag{55}$$

Note that variables x_1, x_2, and x_3 are fixed. Only the variable x_4 is free. In general, for any clause $c \in C$:

$$K(C,f) = K(C', f + f') - K(C'', f + f'') \tag{56}$$

where C' is formed by removing clause c from C, f' is the number of variables in c that are not in any other clause of C. C'' is formed from C and f'' is determined by the procedure:

1. Remove clause c from C.
2. Remove every clause that contains the negation of any term in c. Set f'' to the number of variables not in c that this procedure removes from the CNF, i.e. these variables are free.
3. In the remaining clauses, remove every occurence of any term that also occurs in c. If this leaves empty clauses, remove them.
4. Merge duplicate clauses.

The base case of recursion 56 occurs when C is the empty CNF. In this case, $K(C,f) = 2^f$.

Since the recursion results in CNF problems at every level, special properties of CNF problem instances can be used to evaluate the subproblems. For example, if the CNF C in $K(C,f)$ can be separated into two sets of clauses that share no variables, then the sets of clauses form two independent CNF expressions. The product of the number of satisfiers of the first expression with the number of satisfiers of the second is the number of satisfiers of the original CNF. For example:

$$K((x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (x_4 \vee \overline{x_5}) \wedge (x_4 \vee x_6), f) \tag{57}$$

$$= K((x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3), f) \times K((x_4 \vee \overline{x_5}) \wedge (x_4 \vee x_6), 0) \tag{58}$$

Note that the free variables are not duplicated among the subproblems. Duplicating the free variables would be equivalent to doubling the number of free variables, i.e. the number of satisfiers of the original CNF would be incorrectly multiplied by $(2^f)^2$ instead of 2^f.

6.5.4 Permanent

If Z is a set of columns that contains every nonzero element in some row, then Z is a zero set.

Let C_i be the set of columns with nonzero elements in row i. By the completer reduction, if one column r of C_i is in R and the rest are in S, then $N(R,S) = N(R - \{r\}, S)$. So choose the split element r in (3) to be a column in $C_i \cap R$ for the $C_i \subseteq R \cup S$ with minimum $|C_i \cap R|$.

Any column of zeroes is vestigial, with $X = \emptyset$. If a matrix has a zero column, then its permanent is zero.

7 Problem Decomposition

Suppose we wish to count the s-t Hamiltonian paths in a graph G containing a cut vertex x. If the removal of x leaves s and t in separate components, then form G_{sx} and G_{xt} as shown in Figure 19. The number of s-t Hamiltonian paths in G is the product of the number of s-x Hamiltonian paths in G_{sx} with the number of x-t Hamiltonian paths in G_{xt}. Computing the number of Hamiltonian paths in G directly requires $O(2^n poly(n))$ time. If G_{sx} and G_{xt} each contain about half as many vertices as G, then computing the number of Hamiltonian paths in both G_{sx} and G_{xt} requires $O(2^{\frac{n}{2}} poly(\frac{n}{2}) + 2^{\frac{n}{2}} poly(\frac{n}{2})) = O(2^{\frac{n}{2}} poly(\frac{n}{2}))$. This is roughly the square root of the time required for direct computation!

Decomposition is more complicated for the subproblems generated by the $N(R,S)$ recursion. Recall that $N(R,S)$ is the number of length $n+1$ s-t walks in $G(V-S)$ that contain every vertex in R. These walks may snake back and forth across the cut vertex, and the length of the subwalks in $G(V-S)_{sx}$ and $G(V-S)_{xt}$ may vary. So $N(R,S)$ cannot be computed by taking the product of a population of s-x walks in $G(V-S)_{sx}$ with a population of x-t walks in $G(V-S)_{xt}$. Instead, the walks of $N(R,S)$ can be counted through partitioning them by (1) the number of occurrences of x, and (2) the total lengths of the subwalks in each of $G(V-S)_{sx}$ and $G(V-S)_{xt}$.

Let $N_{sx}(R_{sx},S,k,c)$ be the number of length k s-x walks in $G(V-S)_{sx}$ that contain every vertex in R_{sx}, and that have $c+1$ occurrences of vertex x. Let $N_{xt}(R_{xt},S,k,c)$ be the number of length k x-t walks in $G(V-S)_{xt}$ that contain every vertex in R_{xt}, and that have $c+1$ occurrences of vertex x. Each walk counted by $N_{sx}(R_{sx},S,k,c)$ consists of an s-x subwalk, followed by c x-x subwalks. Each walk counted by $N_{xt}(R_{xt},S,k,\hat{c})$ consists of $\hat{c}$ x-x subwalks, followed by an x-t subwalk. Each N_{sx} walk can be combined with each N_{xt} walk to form a set of s-t walks by merging the sequence of c N_{sx} x-x subwalks with the sequence of $\hat{c}$ N_{xt} x-x subwalks in $C(c+\hat{c},c)$ ways. The length of the combined walks is the sum of the lengths of the N_{sx} and N_{xt} walks, and the combined walks contain every vertex in $R_{sx} \cup R_{xt}$, so:

$$N(R,S) = \sum_{k=1}^{n}\sum_{c=1}^{n}\sum_{\hat{c}=1}^{n} C(c+\hat{c},c)N_{sx}(R_{sx},S,k,c)N_{xt}(R_{xt},S,(n+1)-k,\hat{c}) \quad (59)$$

Further reasoning about the limits of summation yields:

$$N(R,S) = \sum_{k=|R_{sx}|}^{(n+1)-|R_{xt}|}\sum_{c=1}^{\lceil \frac{k}{2} \rceil}\sum_{\hat{c}=1}^{\lceil \frac{(n+1)-k}{2} \rceil} C(c+\hat{c},c)N_{sx}(R_{sx},S,k,c)N_{xt}(R_{xt},S,(n+1)-k,\hat{c}) \quad (60)$$

$N_{sx}(R_{sx},S,k,c)$ and $N_{xt}(R_{xt},S,k,c)$ can be computed recursively:

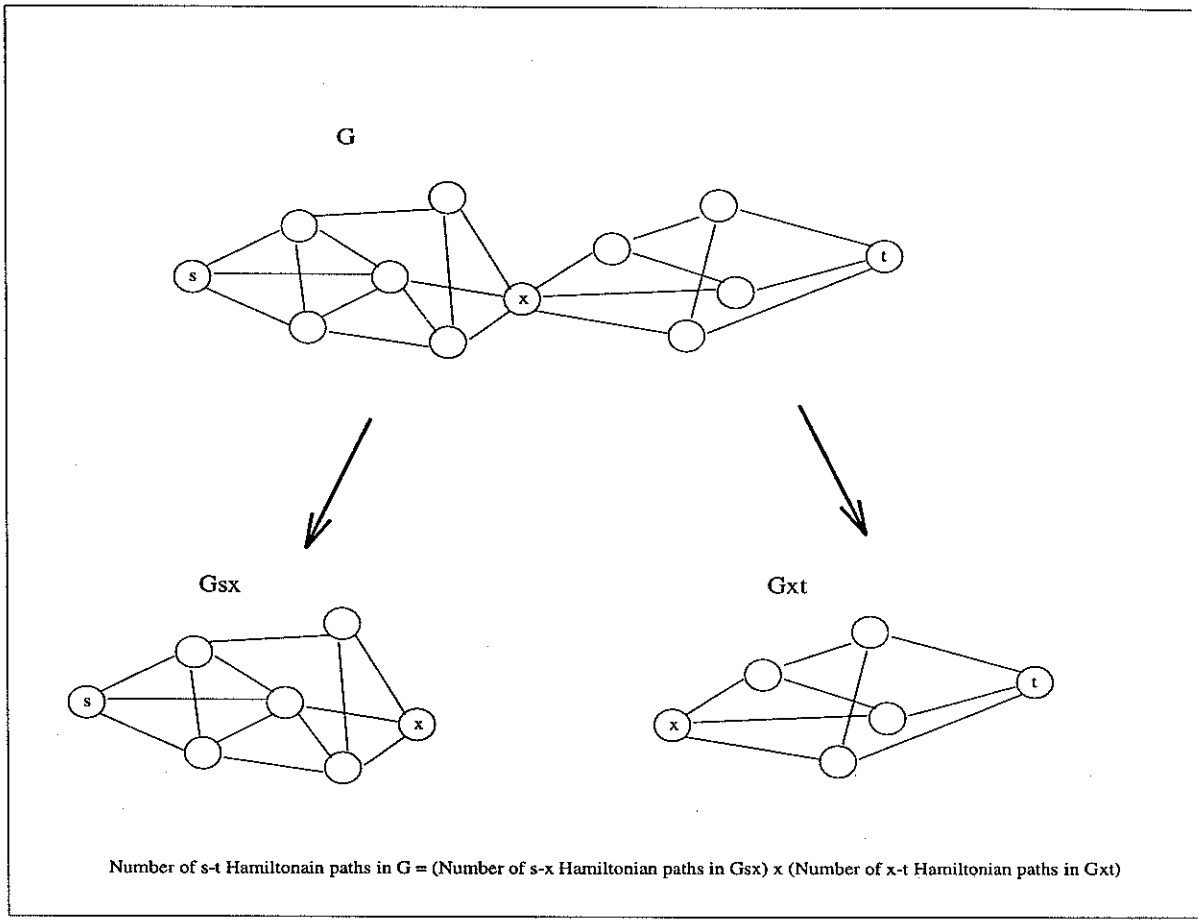


Figure 19: s-t Cut Vertex Decomposition of the Hamiltonian Path Problem

$$\forall r \in R\ N_{sx}(R_{sx}, S, k, c) = N_{sx}(R_{sx} - \{r\}, S, k, c) - N_{sx}(R_{sx} - \{r\}, S \cup \{r\}, k, c) \tag{61}$$

and

$$\forall r \in R\ N_{xt}(R_{xt}, S, k, c) = N_{xt}(R_{xt} - \{r\}, S, k, c) - N_{xt}(R_{xt} - \{r\}, S \cup \{r\}, k, c) \tag{62}$$

The base case occurs when $R_{sx} = \emptyset$ or $R_{xt} = \emptyset$. Then N_{sx} or N_{xt} can be computed by an algorithm similar to the algorithm to count $N(\emptyset, S)$ in the original Hamiltonian path counting problem.

If $G(V - S)$ has a cut vertex x that does not separate s from t, then there is a decomposition similar to 60:

$$N(R,S) = \sum_{k=|R_{st}|}^{(n+1)-|R_{xx}|} \sum_{c=1}^{\lceil \frac{k}{2} \rceil} \sum_{\hat{c}=1}^{\lceil \frac{(n+1)-k}{2} \rceil} C(c+\hat{c}, c) N_{st}(R_{st}, S, k, c) N_{xx}(R_{xx}, S, (n+1)-k, \hat{c}) \tag{63}$$

where N_{st} and N_{xx} are analogous to N_{sx} and N_{xt} – Nst counts s-t walks in the biconnected component containing vertices s and t, while N_{xx} counts x-x walks in the other biconnected component.

Decomposition of the Hamiltonian path counting problem creates new types of subproblems, which introduces the challenges of finding reductions and decompositions for the subproblems. The basic $N(R, S)$ reductions apply automatically, but identifying symmetries, zero sets, and vestigial elements for the new subproblems requires some thought.

Not all decompositions are so complicated. Since the $N(R, S)$ recursions for counting vertex colorings and counting CNF satisfiers have been reduced to recursions that yield instances of the original problem as subproblems, decomposition has been reduced to identifying instances which can be partitioned into independent instances of the original problem. For counting vertex colorings, if the graph is disconnected then each component can be colored separately. For counting CNF satisfiers, if the CNF expression can be partitioned into sets of clauses that share no variables, then the satisfiers of the sets of clauses can be computed separately.

For the permanent, decomposition of $N(R, S)$ instances defined on the original matrix generates $N(R, S)$ instances defined on submatrices. Consider the matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{64}$$

$N(\{1,4,5\},\{3\})$ is the number of row terms with elments from columns 1, 4, and 5 and with no elements from column 3. Since row terms with elements from column 3 are not counted, $N(\{1,4,5\},\{3\})$ is the same if the third column of the matrix is zeroed:

$$\hat{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{65}$$

In $\hat{A}$, every nonzero element in column 1 or 2 is in row 1 or 2. Also, every nonzero element in column 4 or 5 is in row 3, 4, or 5. Define a P row term to be a set of $|P|$ nonzero elements that contains one element from each each row indexed by P. Every row term in $\hat{A}$ can be partitioned into a $\{1,2\}$ row term with elements only from the first two columns and a $\{3,4,5\}$ row term, with elements only from the last two columns. Thus, $N(\{1,4,5\},\{3\})$ is the product of the number of $\{1,2\}$ row terms that contain at least one element from column 1 with the number of $\{3,4,5\}$ row terms that contain at least one element from each of the columns 4 and 5. Recall that in the permanent computation, $N(R,S)$ is the number of row terms that contain elements from every column indexed by R and that contain no elements from any column indexed by S. Now:

$$N(\{1,4,5\},\{3\}) = N_1(\{1\},\emptyset) \times N_2(\{4,5\},\emptyset) \tag{66}$$

where $N_1(\hat{R},\hat{S})$ is $N(R,S)$ restricted to the submatrix induced by row set $\{1,2\}$ and column set $\{1,2\}$, and $N_2(\hat{R},\hat{S})$ is $N(R,S)$ restricted to the submatrix induced by row set $\{3,4,5\}$ and column set $\{4,5\}$.

To decompose a given $N(R,S)$ instance, partition the rows into sets $P_1,\ldots,P_K$ and partition the columns not indexed by S into the corresponding sets $Q_1,\ldots,Q_K$, such that for each k, every nonzero element in the rows indexed by P_k is in one of the columns indexed by Q_k, and every nonzero element in the columns indexed by Q_k is in one of the rows indexed by P_k. (In the previous example, $P_1 = \{1,2\}$, $Q_1 = \{1,2\}$, $P_2 = \{3,4,5\}$, and $Q_2 = \{4,5\}$.) The following procedure produces the row partition $P_1,\ldots,P_K$:

initially $\mathbf{P}=\{\{1\},\ldots,\{n\}\}$
for $c \notin S$
  Merge the sets in $\mathbf{P}$ that index at least one row with a nonzero element in column c.

After the procedure, $\mathbf{P}=\{P_1,\ldots,P_K\}$. (If $|\mathbf{P}| = 1$, then there is no decomposition.) Partition the columns into $Q_1,\ldots,Q_K$ as follows:

$$\forall k \in \{1, \ldots, K\} Q_k = \bigcup_{i \in P_k} \{j | a_{ij} = 1 \wedge c \notin S\} \tag{67}$$

Define $N_k(\hat{R}, \hat{S})$ to be the number of P_k row terms that contain at least one element from each column indexed by $\hat{R}$ and that contain no elements from the columns indexed by $\hat{S} \cup \overline{Q_k}$. Then:

$$N(R, S) = \prod_{k=1}^{K} N_k(Q_k \cap R, \emptyset) \tag{68}$$

The standard recursion holds for $N_k(\hat{R}, \hat{S})$:

$$N_k(\hat{R}, \hat{S}) = N_k(\hat{R} - \hat{r}, \hat{S}) - N_k(\hat{R} - \hat{r}, \hat{S} \cup \hat{r}) \tag{69}$$

and the base case is:

$$N_k(\emptyset, \hat{S}) = \prod_{i \in P_k} \sum_{j \in Q_k - \hat{S}} a_{ij} \tag{70}$$

8 Detection and Bounding Algorithms

The counting algorithm can be altered to solve the detection problem, i.e. to determine whether or not $N(\{1\ldots n\},\emptyset) > 0$. We follow the branch and bound model [13], but we use the method for counting rather than optimization.

The algorithm begins with a single active node representing $N(\{1\ldots n\},\emptyset)$. At each step an active node is replaced by an equivalent set of active nodes (branches) or computed $N(S)$ terms (leaves). Together, the values of the active nodes and the computed $N(S)$ terms always sum to $N(\{1\ldots n\},\emptyset)$. Symmetry is implemented by using a coefficient c to indicate how many duplicate $N(R,S)$ terms each active node represents.

At each step we maintain a lower bound L on $N(\{1\ldots n\},\emptyset)$:

$$L = \sum_{node(R,S,c)\in active} cL(R,S) + \sum_{leaf(S)\in computed} (-1)^{|S|}N(S) \qquad (71)$$

where $L(R,S)$ is a lower bound on the contribution of $N(R,S)$ to $N(\{1\ldots n\},\emptyset)$.

In successive applications of the recursion (3) to $N(\{1\ldots n\},\emptyset)$, $N(R,S)$ terms count positively in the sum if $|S|$ is even and negatively if $|S|$ is odd. Using the bounds given by the nonnegativity theorem and the bounding theorem with $|Q|$=1, we obtain:

$$L(R,S) = \begin{cases} 0 & \text{if } |S| \text{ is even} \\ -\min_{r\in R} N(S) - N(S\cup\{r\}) & \text{if } |S| \text{ is odd} \end{cases}$$

We also maintain an upper bound U:

$$U = \sum_{node(R,S,c)\in active} cU(R,S) + \sum_{leaf(S)\in computed} (-1)^{|S|}N(,S) \qquad (72)$$

where $U(R,S)$ is an upper bound on the contribution of $N(R,S)$ to $N(\{1\ldots n\},\emptyset)$:

$$U(R,S) = \begin{cases} \min_{r\in R} N(S) - N(S\cup\{r\}) & \text{if } |S| \text{ is even} \\ 0 & \text{if } |S| \text{ is odd} \end{cases}$$

If $L > 0$ then $N(\{1\ldots n\},\emptyset) > 0$. If $U = 0$ then $N(\{1\ldots n\},\emptyset) = 0$. When $L = U$, $N(\{1\ldots n\},\emptyset)$ has been exactly determined.

The branch and bound detection algorithm is:

```
detect()
{
function L(R,S), U(R,S), NØ(S);
integer c, L, U;
set R, S, A, M, active;
```

$L = 0$; $U := U(\{1 \ldots n\}, \emptyset)$;
$active = \{node(\{1 \ldots n\}, \emptyset, 1)\}$;

while $(L < U)$
    remove some $node(R, S, c)$ from *active*.
    $L := L - cL(R, S)$; $U := U - cU(R, S)$;
    M:=largest set in the partition of R induced by $s()$;
    if $(|M| = |R|)$ (* Branch to form leaves *)
        for $(i = 0; i \leq |M|; i++)$
            A=first i elements of M;
            c=C$(|M|, i)$;
            $L := L + (-1)^{|S|} cN(S \cup A)$;
            $U := U + (-1)^{|S|} cN(S \cup A)$;
    else (* Branch to form active nodes. *)
        for $(i = 0; i \leq |M|; i++)$
            A=first i elements of M;
            c=C$(|M|, i)$;
            $L := L + cL(R - M, S \cup A)$;
            $U := U + cU(R - M, S \cup A)$;
            if $(L(R - M, S \cup A) < U(R - M, S \cup A))$
                insert node$(R - M, S \cup A, c)$ into *active*;
    if $(L > 0)$ return(true); (* $N(\{1 \ldots n\}, \emptyset) > 0$ *)
    if $(U = 0)$ return(false); (* $N(\{1 \ldots n\}, \emptyset) = 0$ *)
if $(L > 0)$ return(true); else return(false); (* $L = U$. Exact count accomplished. *)
}

In the worst case the detection algorithm mimics the counting algorithm, i.e. every leaf $N(S)$ is computed. Choosing nodes at random from *active* can lead to an exponential number of active nodes. However, always choosing an active node with minimum $|R|$ limits the number of active nodes to $O(n^2)$.

To observe the algorithm's progress to convergence, return the values of the upper and lower bounds U and L after every branching. To stop as soon as $N(\{1 \ldots n\}, \emptyset)$ is known to within k, change "while $(L < U)$" to "while $(U - L \leq 2k)$", remove the last three lines of the algorithm, and insert "return $((U + L)/2)$;" at the end of the algorithm.

9 Open Questions and Future Directions

9.1 Multicomputer Implementation

It would be interesting to experiment with multicomputer implementations of our recurrence-based algorithms. The recursion formula (3) could be the basis for dynamically allocating work among processors to achieve load balance.

It may be efficient to have a base case with $|R| > 0$, i.e. to stop the recursion and directly compute $N(R,S)$ before $R = \emptyset$. For example, consider the s-t Hamiltonian path problem. Recall that $N(R,S)$ is the number of length $n+1$ s-t walks in $G(V-S)$. Let $E(V-S)$ be the edges in $G(V-S)$. Define $w(L,x,M)$ to be the number of length L s-x walks that contain every vertex in $M \subseteq R$. Then $w(n+1,t,R) = N(R,S)$, and we can compute $w(n+1,t,R)$ by the dynamic program:

initially $w(0,s,\emptyset) = 1$
for $L = 1$ to $n+1$
  $\forall\, v \in R, \forall\, M \subseteq R$ with $|M| \leq L-1$
    $w(L,v,M) := \sum_{uv \in E(V-S)} w(L-1,u,M)$
  $\forall\, v \notin R, \forall\, M \subseteq R - \{v\}$ with $|M| \leq L-1$
    $w(L,v,M \cup \{v\}) := \sum_{uv \in E(V-S)} [w(L-1,u,M \cup \{v\}) + w(L-1,u,M)]$

The algorithm requires $O(2^{|R|}|E(V-S)|n)$ time and $O(2^{|R|}n)$ space. Dynamic programming avoids the $O(2^{|R|})$ recursive calls required for the recurrence-based algorithm, but dynamic programming requires exponential space. The most efficient base case size of R depends on the capabilities of individual processors, the overhead cost of recursive calls, and the problem size.

If reductions are rare, or their tests are expensive, then it may be best to search for them intermittedly instead of at every recursive call. The frequency of tests could be some constant, or it could vary dynamically. For example, to test at every third level of recursion, we could test for reductions only if $|R|$ is a multiple of three. Or, in the Hamiltonian path problem, we could test for reductions only when $G(V-S)$ is sparse.

9.2 Symmetry

Symmetry deserves further study. The challenge is to develop efficient tests that examine more cases of symmetry, with emphasis on searching for the most probable symmetries. For example, in our Hamiltonian path algorithm, we test for symmetries in which the underlying automorphism maps many of the vertices to themselves. Intuitively, these seem to be the most likely symmetries. But how likely are they? How unlikely are more complicated symmetries?

When should we expect the search for symmetry to pay off? Obviously, very dense and very sparse graphs are the most likely to have symmetries. But how likely? For example, if a graph has just enough edges that it almost surely has a Hamiltonian cycle, how many symmetric vertices can we expect it to have?

9.3 Combining Recursive Strategies

The $N(R, S)$ recursion splits a CNF instance into smaller CNF instances by pivoting on some clause. A CNF instance can also be split by pivoting on a variable – sum the satisfiers for the CNF in which the variable is assigned T and the satisfiers for the CNF in which the variable is assigned F. For example, the number of satisfiers of:

$$(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (x_2 \vee x_4) \tag{73}$$

is the sum of the number of satisfiers in which $x_1 = T$:

$$(T \vee x_2) \wedge (\overline{T} \vee x_3) \wedge (x_2 \vee x_4) = (x_3) \wedge (x_2 \vee x_4) \tag{74}$$

and the number of satisfiers in which $x_1 = F$:

$$(F \vee x_2) \wedge (\overline{F} \vee x_3) \wedge (x_2 \vee x_4) = (x_2) \wedge (x_2 \vee x_4) \tag{75}$$

Both recursions yield CNF instances at every level, so either recursion can be used on the instances generated by the other. Choosing the best recursion for a given instance is no simple matter. Every clause is a candidate for the pivot clause, and every variable is a candidate for the pivot variable, so there are many options. The variable strategy removes the pivot variable from both both instances in the split, and each clause containing the pivot variable is removed from one of the split instances. The clause strategy produces a less balanced split. The $N(R - \{r\}, S)$ instance lacks only the pivot clause. The $N(R - \{r\}, S \cup r)$ instance lacks the pivot clause, lacks every variable in the pivot clause, and lacks every clause that contains the negation of any term in the pivot clause. There may be no simple rule to determine the best strategy for a given instance. It may be useful to use a lookahead procedure – something like a game tree search. For each instance, several levels of recursion are applied; each strategy yields a set of CNF instances. The simplest set of CNF instances is chosen (by some metric), and then the same procedure determines the recursion strategy for each instance in the set, and so on, until base cases are produced.

References

[1] E. BAX, Inclusion and exclusion algorithm for the Hamiltonian path problem, *Inf. Proc. Lett.*, 47 (4) (1993), pp. 203-207.

[2] E. BAX, Counting paths and cycles, *Inf. Proc. Lett.*, 52 (1994), pp. 249-252.

[3] R. BELLMAN, Combinatorial processes and dynamic programming, in: R. Bellman and M. Hall, Eds., *Combinatorial Analysis*, Proc. AMS Symposia on Applied Mathematics, Vol. X, American Mathematical Society, Providence, RI, 1960.

[4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability - A Guide to the Theory of NP-Completeness* W. H. Freeman and Company, New York, 1979.

[5] Y. GUREVICH AND S. SHELAH, Expected computation time for Hamiltonian path problem, *SIAM J. Comput., 16 (3) (1987)*, pp. 486-502.

[6] M. HELD AND R. M. KARP, A dynamic programming approach to sequencing problems, *J. Soc. Indust. Appl. Math., 10 (1962)*, pp. 196-210.

[7] R. M. KARP, Reducibility among combinatorial problems, in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

[8] R. M. KARP, Dynamic programming meets the principle of inclusion and exclusion, *Operations Research Letters*, 1 (2) (1982), pp. 49-51.

[9] W. X. LI AND F. TIAN, Some notes on the chromatic polynomials of graphs (Chinese), *Acta Math. Sinica*, 21 (1978), pp. 223-230.

[10] N. LINIAL AND N. NISAN, Approximate inclusion-exclusion, *Combinatorica*, 10 (4) (1990), pp. 349-365.

[11] G. H. J. MEREDITH, Coefficients of chromatic polynomials, *J. Combinatorial Theory (B)*, 13 (1972), pp. 14-17.

[12] A. NIJENHUIS AND H. S. WILF, *Combinatorial Algorithms For Computers and Calculators* Academic Press, New York, 1978.

[13] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization, Algorithms and Complexity* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982, pp. 433-448.

[14] L. PÓSA, Hamiltonian circuits in random graphs, *Discrete Mathematics*, 14 (1976), pp. 359-364.

[15] R. C. READ AND W. T. TUTTE, Chromatic Polynomials, *Selected Topics in Graph Theory 3*, L. W. Beineke and Robin J. Wilson, ed., Academic Press Limited 1988, Ch. 2.

[16] H. J. RYSER, *Combinatorial Mathematics* The Mathematical Association of America, 1963, Ch. 2.

[17] E. SHAMIR, How many random edges make a graph Hamiltonian? *Combinatorica*, 3 (1) (1983), pp. 123-131.

[18] R. TARJAN, Enumeration of the elementary circuits of a directed graph, *SIAM J. Comput.*, 2 (3) (1973), pp. 211-216.

[19] L. G. VALIANT, The complexity of computing the permanent, *Theoret. Comput. Sci.*, 8 (2) (1979) 189-201.

[20] L. G. VALIANT, The complexity of enumeration and reliability problems, *SIAM J. Comput.*, 8 (3) (1979) 410-421.

[21] H. WHITNEY, A logical expansion in mathematics, *Bull. Amer. Math. Soc.*, 38 (1932), pp. 572-579.

[22] D. ZEILBERGER, Garsia and Milne's bijective proof of the inclusion-exclusion principle, *Discrete Mathematics*, 51 (1984), pp. 109-110.